European Sixth Framework Network of Excellence FP6-2004-IST-026854-NoE

Deliverable D7.2

The EMANICS Consortium

Caisse des Dépôts et Consignations, CDC, France
Institut National de Recherche en Informatique et Automatique, INRIA, France
University of Twente, UT, The Netherlands
Imperial College, IC, UK
International University Bremen, IUB, Germany
KTH Royal Institute of Technology, KTH, Sweden
Oslo University College, HIO, Norway
Universidat Politecnica de Catalunya, UPC, Spain
University of Federal Armed Forces Munich, UniBwM, Germany
Poznan Supercomputing and Networking Center, PSNC, Poland
University of Zürich, UniZH, Switzerland
Ludwig-Maximilian University Munich, LMU, Germany
University of Surrey, UniS, UK
University of Pitesti, UniP, Romania

For more information on this document or the EMANICS Project, please contact:

Dr. Olivier Festor
Technopole de Nancy-Brabois — Campus scientifique
615, rue de Jardin Botanique — B.P. 101
F—54600 Villers Les Nancy Cedex
France

Phone: +33 383 59 30 66
Fax: +33 383 41 30 79
E-mail: <olivier.festor@loria.fr>

Document Control

Title: Final report on benchmarking of management protocols

Type: Public

Editor(s): Radu State

E-mail: state@loria.fr

Author(s): Laurent Andrey, Juergen Schoenwaelder, Aiko Pras, George Pavlou, Krzysztof Nowak, Tudor Balanescu

Doc ID: D7.2-v0.1.doc

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
V0.1	April 2007	Radu State	First version
V0.2	May 2007	Laurent Andrey	Added JMX model impact section. - Overall formatting
V0.3	July	Radu State	Content add and formatting

Legal Notices

Table of Contents

(This page is left blank intentionally.)

Executive Summary

This deliverable summarizes key research results obtained during the second phase of the WP7 activities. The research activities addressed the performance evaluation of management frameworks (SNMP and JMX). For these management frameworks, the current work presents qualitative and quantitative requirements associated to the management task. The SNMP framework is analysed with respect to a set of traces gathered from an operational network. The JMX framework is analysed based on synthetic data obtained from a process injection driven activity.

Introduction

This document describes experiments and benchmarking performed on SNMP and JMX management framework as well as specific requirements for monitoring large scale network. The structure of this document is as follows: Section 2 describes the requirements and monitoring architectures for large scale networks. Section 3 addresses the results and approaches developed for the analysis of SNMP traces obtained for multiple networks. A specific case based study is developed in section 4 and covers the measurements and in depth analysis of the PIONEER infrastructure operated by PSNC. A final section 5 covers the experimental results and analysis for the benchmarking of the JMX framework.

1 Network Monitoring in large scale networks

Monitoring of network status and its resources is a required process in order to ensure that the network operation will not be inhibited and that any network will run smoothly. This is even more important when providing Quality of Service based value-added services (QoS) across a network. Such a task as that of providing QoS based services is very challenging, and requires the employment of resource management techniques such as the use of traffic engineering. Traffic engineering though relies on the collection of various monitoring data in order to perform both proactive and dynamic reactive solutions that will ensure the smooth operation of a network. Even though it becomes clear that Quality of service (QoS) monitoring is a very important aspect in order to provide quantified QoS-based services and assure to subscribing customers that the objectives of their contract are being met, it is not an easy task. The reason for this is that Traffic engineering (TE) sometimes have requirements such as maximizing network resource utilization and at the same time meet the QoS demands of services contracted to customers that can be difficult to achieve. To be able to measure, characterize and control traffic which are some of the goals of TE it is imperative to have knowledge of the network status and the history for the decisions taken. Such knowledge can be provided by measurements, where the operational state of the network is monitored.

Currently QoS is provided on the basis of Service Level Agreements (SLAs). These represent a set of terms that clients and providers of services have to abide by when they are accessing, providing a service respectively. The technical part of an SLA is called a Service Level Specification (SLS) and it represents the means for defining QoS-based IP connectivity services [1]. IP Differentiated Services [2] (DiffServ) is currently seen as the framework for providing services across the internet. In this model of offering services, routers aggregate traffic that belong to several service classes according to predefined QoS policies. These policies are quantified by means of using performance parameters such as throughput, delay, loss and delay variation. As the network attempts to offer several different classes of services, network and service monitoring are becoming more important in order to provide service assurance to customers.

To collect though such a variety of management data the deployment and use of a proper QoS monitoring infrastructure is required. By means of proper, a monitoring system should be scalable in terms of network size, speed, and number of customers that will consume any offered services. Given the multitude of services with different performance requirements and the need to have measurement data with the finer granularity possible the design and implementation of *scalable* monitoring systems capable of providing measurements for network provisioning, dynamic resource allocation, route management, and in-service verification of value-added services is a big challenge.

1.1 Monitoring Systems Review

There have been many working groups in the Internet Engineering Task Force (IETF) that have been carrying out work on monitoring and measurements. One of these groups is the IP Performance Metrics (IPPM) [3] working group that has defined metrics as the one-way delay, loss, delay variation, round-trip delay, and link bandwidth that can be used to measure the quality, performance, and reliability of a data delivery path.

Another group is the IETF IP Flow Information Export (IPFIX) [3] working group which defined a system architecture for exporting IP flow information. Other groups work such as the IETF Real-Time Traffic Flow Measurement (RTFM) [3] defined an architecture and methodology for measuring real time information. Though this groups' work found implementers because of its powerful and flexible characteristics, the implementations were very complicated and difficult to use and as such the group's work became obsolete. The IETF Packet Sampling (PSAMP) [3] working group was chartered to define a set of standard capabilities for sampling packets through statistical and other methods in order to support measurements in high speed networks. Through packet sampling techniques a representative subset of packets is selected and used to provide knowledge about the status of the whole set of observed packets without processing them all.
Many software tools based on this work have been implemented to provide path performance and network-wide measurements. The RIPE Test Traffic Measurement (TTM) [4] project provides capabilities to measure one-way delay, packet loss, bandwidth etc between installed measurement probes. NetFlow from Cisco provides data filtering and aggregation capabilities using a flow analyzer to report statistics about micro or aggregated flows. The National Internet Measurement Infrastructure (NIMI) [5] is a system that uses servers that are deployed between specifically chosen network points to compute traffic characteristics based on active probes. Other groups such as the Measurement and Network Analysis Group of the National Laboratory for Applied Network Research [6], the Cooperative Association for Internet Data Analysis (CAIDA) [7] and the Sprint IP Monitoring (IPMON) [8] project have defined a big variety of tools to address the problem of monitoring internet traffic.

Most of the research we have mentioned so far did not take any special consideration of the issues arising when QoS guarantees are offered when crossing multiple domains [9]. Such work on monitoring and measurements for traffic engineering has been performed by some European research projects. One such project is the IST-INTERMON [9] aiming at developing an integrated inter-domain QoS monitoring, analysis, and modeling system that can be used for planning, operational control, and optimization. The goal of other projects like the IST-MoMe [9] project is to provide inter-domain real time QoS architectures with integrated monitoring and measurement capabilities. Additionally the IST-SCAMPI [9] project tries to design an extensible network monitoring architecture that will provide monitoring capabilities even at 10 Gbps speeds. The latter project also works on several other measurement tools to can be used for addressing other aspects of network management such as denial of service detection, accounting etc.

Collecting, analyzing, and visualizing Internet traffic data such as network topology, traffic load, performance, and routing parameters is not without any consequences. When the size of the network being monitored grows, especially for large IP networks a large amount of information needs to be collected especially when dealing with networks used for providing QoS guarantees since TE and service-level monitoring needs to be performed. As such designing a scalable monitoring system is a very big challenge and requires special consideration in the formation of its architecture. In the next sections we propose a set of principles to design a scalable monitoring system and we assess its performance. This monitoring system covers the needs of a framework in terms of short or long term data for providing QoS guarantees over an MPLS backbone. The system supports QoS-based service level monitoring and provides QoS-based real-time monitoring support for TE in MPLS DiffServ-capable networks. To assess the

performance of this system based on these principles we analyze the requirements of this system in terms of the measurements that need to be performed. Based on these requirements we give details on how to perform them and we also provide some experimental results of the performance of the monitoring system.

1.2 Monitoring guidelines and principles

Providing QoS over MPLS-enabled networks in a single domain or across different domains has been or is the focus of the study of many researchers. Examples of such work is the TEQUILA [11] architecture, the CADENUS architecture [12], the ENTHRONE architecture [13] etc. All of these architectures rely on their monitoring system to provide them with up to date information about the state of their network in order to provide QoS guarantees to their clients. To address the needs of traffic engineering and QoS provisioning in general, a monitoring system must acquire long and short term data. The fact though that QoS enabled networks may have a large number of nodes and a large number of routes with different QoS guarantees may result in an exponential increase of monitoring requirements. As such some of the aforementioned architectures propose the following for their monitoring system [10] [13]:

- In Diffserv traffic engineered networks a number of different classes of service exist. Therefore apart from the IP routing protocol for the establishment of IP routes, the Multi-Protocol Label Switching (MPLS) protocol is required for the establishment of explicit paths called Label Switched Paths (LSPs). These paths carry aggregate traffic belonging to different SLSs with similar performance requirements. Usage of TE algorithms to carry on packet-level micro-flow statistics would be very expensive. Instead statistics can be gathered on the aggregate macro-flow level. Thus monitoring should be performed on the basis of the class of service defined and offered between an ingress router and an egress router. As such measurements are only performed at the level of PHBs and LSPs..
- The monitoring and data collection systems should be distributed and close to the source to minimize traffic requirements and processing overhead.
- Since SLSs belong to different classes of service they have different requirements. A premium class needs more measurements than a best effort service. Using different sampling frequencies for SLSs would make the monitoring architecture more complex. SLSs monitoring based on aggregate performance measurements (delay, jitter, loss) is more scalable since one measurement can satisfy many SLSs on the same route. Therefore data referring to aggregate information can be combined with traffic contract measurements (Service Level Specifications - SLSs), since one measurement may satisfy many SLSs.
- Hop by Hop measurements can be used to compute traffic from edge to edge for low profile classes since the accuracy of the hop by hop technique may not be as precise as required for higher profile traffic classes. Edge-to-edge measurements are very expensive for a fully meshed N-node network, since more than N^2 LSPs must be monitored (load balancing uses multiple routes). A hop-by-hop approach where addition of the hop-by-hop measurements is performed to calculate the edge-to-edge result is more scalable.

To fulflll the above requirements a monitoring system must be able to perform two kinds of measurements; active and passive ones. Active measurements are performed by injecting synthetic traffic to the network in order to monitor delay, jitter and packet loss. Passive measurements can be conducted with SNMP and involve measuring throughput, load and packet discards at the PHB, SLS and LSP level.

1.3 Passive monitoring

A monitoring system should be able to perform proper load balancing of the network or identify violations of traffic contracts by the customer or the network configuration in order to provide QoS guarantees to each customer. To do that the monitoring system should be able to capture-measure aspects of the network such as the following [10]:

- LSP load at the ingress router (LSP L-I)
- LSP throughput at the egress router (LSP T-E)
- PHB throughput at every router (PHB T)
- PHB packet discards at every router (PHB D)
- Offered load at ingress per SLS or flow (SLS L)
- Offered through-put at egress per SLS or flow (SLS T)

To perform these measurements using the Simple Network Management Protocol (SNMP), special Management Information Bases (MIBs) must be used. The first five of the six measurements above can be performed by using two of SNMP's MPLS MIBs, the MPLS Label Switching Router (LSR) MIB (RFC 3813) [14] and the MPLS Forwarding Equivalence Class to Next Hop Label Forwarding Entry (FEC-To-NHLFE) MIB [15] (RFC 3814). The LSR MIB can be used to perform PHB and LSP measurements and the FEC MIB can be used to perform the SLS load measurements. The sixth measurement cannot be performed with the MPLS MIBs because the FEC-to-NHLFE MIB is intended to be deployed only at the ingress router. Special tools such as Netflow need to be used to perform this kind of measurement

1.3.1 Passive monitoring process description

In order to perform the first five passive measurements using SNMP, different tables and entries and MIBs must be accessed. The operations for the acquisition of data from these tables are based on the manager–agent model. For LSP measurements the manager must query the agent for the following data:

- LSP-IDs in order to determine from the relevant table how LSPs are organized in it. Also in this step the row identifiers (row IDs) to be able to perform the next step are returned.
- LSP load in the ingress (LSP L-I) or throughput (LSP T-E) at the egress from tables holding statistical information using the row-IDs in the previous step .

For PHB measurements the process is the following:

- Retrieve the PHB IDs to determine to which PHB each LSP belongs to.
- Then having the row IDs from the previous query, query for throughput (PHB T) or packet discards (PHB D) for each PHB.

To determine SLS load, the manager must do the following:

- Query to which LSPs each SLS is bound to (SLS IDs, Differentiated Service Code Point field (DSCP) and LSP IDs).

- Then with the row IDs obtained, the manager can access the statistics table for each LSP to compute the load for each traffic contract (SLS).

2.3.2 Passive monitoring details granularity and assumptions

To explain why in the monitoring process we need to determine first the IDs of LSPs, SLSs and PHBs, we need to state some assumptions. For our measurements we assume that the manager knows about the topology of the network [10]. This means that the manager has knowledge of the ID of every contract, the IDs of each LSP in the ingress and egress router and the traffic classes that exist. Still the manager is not aware in what order this data appear in the relevant tables. This is important because the queries for data are based solely on their position in the relevant tables. As such, the manager needs first to know how LSP PHB and SLS data are organized in the relevant tables and then having this information it can query for load etc from the statistical tables.

Another aspect that needs special consideration is the sampling granularity for each type of data. Determining, for example the IDs for each LSP requires a big sampling period (long granularity) since all LSPs, even the ones used for load balancing, need to be configured only once for long time periods. This is true because reconfiguration of the network to provide QoS guarantees to customers does not occur often. As such relevant data in the ingress or egress router referring to LSP IDs change infrequently. Tabular objects though referring to PHB and SLS specific data on the other hand may change more frequently. This occurs because failures on MPLS-capable interfaces can occur. Solving a problem like the failure of an interface might be possible without reconfiguring the entire network by routing traffic mapped on the failing interface to another. Thus objects referring to specific PHBs or SLSs might change. Therefore the polling period for PHB and SLS data is in the order of 5-10 minutes (medium granularity). Lastly, sampling periods for data such as load, throughput etc for SLSs, PHBs or LSPs are in the order of 10-20 secs (short granularity).

1.4 Active monitoring

There are two distinct methods that can be used to perform the active performance measurements (a) edge-to-edge and (b) hop-by-hop [10]. The former method involves capturing an aspect of the network by injecting synthetic traffic between two edge nodes. The latter method uses measurements of synthetic traffic between all neighboring nodes in the edge to edge path to compute the network metric that needs to be captured. Both of these methods can be used to determine the status of the attached links, interfaces and associated queues. Monitoring scalability though could be a serious concern especially when using an MPLS based network for TE. If for example a full mesh logical network path with N edge nodes is in place, an order of $O(N2)$ unidirectional LSPs need to be monitored. Practically more LSPs will need to be monitored because (a) more LSPs are usually established between edge nodes for load balancing and (b) different services use frequently different LSPs between an ingress-egress node pair. LSP monitoring is scalable and feasible with the edge to edge method only when a limited number of LSPs are going to be selected for edge to edge measurements. For the edge to edge method an active monitoring agent is used to inject synthetic traffic and directly provides edge-to-edge measurement results. On the other hand the hop-by-hop method overcomes the scalability problem of the edge to edge approach by using per service class or PHB level measurements to calculate the

edge-to-edge result. This happens because usually many different edge-to-edge paths are passing through the same link and as such they share the resources of the traffic class on a single interface. This way when synthetic traffic is produced to measure a certain attribute on a particular interface, this measurement also satisfies the monitoring requirements of all the paths that are using the particular hop. This way the synthetic traffic injected in the network is significantly reduced. Apart from the scalability benefit, the hop by hop approach also offers us one more advantage. It allows us to locate in a straightforward way, when a problem such as increased delay or loss arises in the network, the interfaces that contribute in this problem.
To calculate edge to edge delay with the hop by hop method requires adding the delay measurements of every hop. To calculate the packet loss ratio of the edge to edge path requires multiplication of the loss measurements of every hop. For the latter statement to stand we assume that packet losses are independent and not correlated. This may not be true in some cases. This happens because every loss measurement involve events happening in different probability spaces and thus the assumption of the events being independent might not hold if for example the losses occurred because of an overload in traffic that could cause correlated losses. Also for some QoS metrics, such as delay adding hop-by-hop measurements to calculate the edge to edge result may not be absolutely accurate [17]. In these cases the computation of the end-to-end result is only an estimation of the actual end-to-end value. In many cases the estimation error is not analytically quantifiable, which means that the end-to-end composed metric is not the same as the actual end-to-end one [17]. In these cases the hop-by-hop measurements can still be used to compute the edge-to-edge result under two assumptions (a) Either the composition error is negligible and thus the edge-to-edge result computation is a good approximation of the actual value (b) the network administrator could run benchmarking experiments to empirically deduce the difference of the two measurements, and then use these results to infer the actual edge-to-edge result from the composition of the hop-by-hop measurements. Other alternatives may also exist to the ones that are given. In later experiments it is shown that that the difference between the two measurement methods is negligible and dependent mainly on the number of hops in the path.

1.5 Monitoring Architecture

In order to perform the active and passive measurements we have described in the two previous sections a monitoring system has been designed [19]. This system is based on the manager-agent paradigm (NetMon-NodeMon). The interactions between the different components of the system are given in figure I. In this figure the functionality of the monitoring system components is the following [10]

- **Node monitor** (NodeMon). This component is responsible for performing node-related measurements. Only one NodeMon is applied per router. NodeMons are hosted outside of the routers they are attached to on dedicated computers since the ability to perform the required measurements is limited in currently available commercial routers. NodeMons can perform both active at the path or hop level, as well as passive monitoring of the router to which they are attached to.
- **Network monitor** (NetMon). This component is the actual manager of the monitoring systems and is capable of performing network-wide post-processing of measurement data using statistical functions. NetMon utilizes network-wide performance and traffic measurements that it collects by all the NodeMon components. This way it can build a physical and logical view of the network by

finding out the routes that have been established. The data this component collects and analyzesare used for non-real-time proactive control of the network by other components

- **Monitoring repository** (MonRep). This component consists of a *data store* which is actually a database for storing large amounts of data from monitoring components, and an *information store* for storing other data of different nature such as configuration type information, information about active monitoring processes etc. Measurement data stored in the data store are used by other processes for TE and service provisioning.
- **Topology repository** (MonRep). This component contains information about the topology of the managed network, LSP related information (LSP IDs, LSP ingress, LSP egress), PHB related information (available traffic classes, traffic classes characteristics etc) and SLS related information (DSCP tags, LSPs bound to an SLS). Information in this component are updated dynamically.

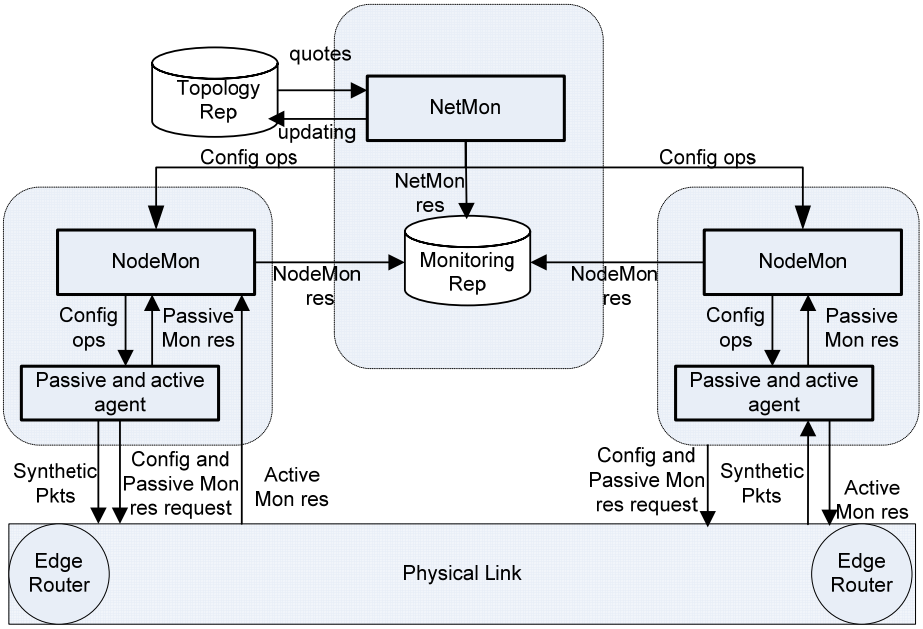


Figure 1.1 *Monitoring System interactions*

For passive monitoring the data that need to be acquired, the process to acquire them, the interactions between different components, the granularity of the measurements the assumptions that are made etc are described in the passive monitoring section. The proposed Monitoring system provides passive traffic measurements at the aggregated or micro/macro-flow level (e.g., per SLS, PHB, LSP) granularities. Core routers collect data on an aggregated form at the PHB level. Ingress/egress routers collect data at the aggregated or micro/macro-flow level. Traffic data are collected in the routers and are passed to the attached NodeMons to be analyzed further. For traffic measurements, continuous per-hop passive traffic monitoring is required for resource management and to guarantee that contracts are being met. In this kind of monitoring we rely on the measured information and the granularity it is obtained from routers. When the speed of the network increases micro flow measurement at the ingress/egress nodes may not be possible. In this case appropriate sampling is performed to keep up with the increasing speed of the network.

Because Core routers are only involved in monitoring at the aggregated level there is no scalability concern.

For performance measurements (delay, loss, jitter), synthetic traffic is injected into the network and then its behavior is captured. The type of synthetic packets sent between two nodes (NodeMons) is the Type-P synthetic packets in [17]. The time difference between the NodeMon-receiver timestamp and the NodeMon-sender timestamp allows us to calculate one-way delay. Because the synthetic packets also have a sequence number one-way packet loss can be calculated from the packets that fail to arrive at the destination NodeMon in a time that is reasonable and represented by a threshold. To measure packet delay variation (ipdv, jitter) the difference of one-way delay between selected packets need to be captured. Additionally it is also possible to measure delay variation as the absolute value of the difference between the time it took two subsequent synthetic packets to reach the same destination point having the same starting point.

Because performance measurements involve time, and they are collected in the hop-by-hop approach from distributed measurement points in the hop by hop path, they must be carried out in a relatively synchronized manner. This means that the traffic flow in order to be monitored appropriately must been “seen” at the same time interval at each hop along the path. Therefore, to achieve greater accuracy in the per-hop approach the nodes along the monitored path must be relatively synchronized so that the extrapolation performed to calculate the edge-to-edge result yields a value close to the result that the edge-to-edge method would give. Therefore, mechanisms for performing, recording, retrieving, and consolidating measurement information are needed to ensure the integrity and validity of measurements [19].

1.6 Evaluation

In this section the performance of the monitoring system in terms of measurement accuracy and scalability, and the ability to support per traffic class traffic engineering is tested. To produce the experimental results for this evaluation a test-bed (Figure II) consisting of four commercial routers connected through three 2 Mb/s serial links in a linear fashion is used. All routers have DiffServ capabilities for traffic classification, traffic conditioning, and various scheduling disciplines. As far as the scheduling discipline, Low Latency Queuing (LLQ) together with Class-Based Weighted Fair Queuing (CBWFQ) are used. LLQ supports strict priority queuing. CBWFQ assigns a queue for each traffic flow and handles each queue in such a way as to ensure that each queue receives only the right amount of available bandwidth during congestion. This is performed by assigning a weight on each flow so as to share the bandwidth in such a way so that high-priority flows are treated in a favored way compared with low profile flows. CBWFQ also supports user-defined traffic classes. CBWFQ allows users to define the exact amount of bandwidth that needs to be assigned to each class of services.

For the NodeMons attached in the routers of the test-bed, Pentium 1.5 GHz PCs are used. Each PC hosts the node monitoring software with the exception that ER1 also hosts the network monitoring software. Usually NetMon is a manager that is located inside a network management centre but for convenience in our experiments it is attached in edge router ER1. To simulate delays and to produce loss events which are mutually independent two Data Channel Simulators (DCSs) are used on link 1 and 3. Synthetic Traffic is produced using a commercial traffic generator that is attached to edge routers ER1 and ER2. We verify whether the measured delay and loss is equal to

the ones simulated by the DCSs by comparing the relevant measured metrics with the ones simulated. As previously mentioned since measurements involve time synchronization we used an accurate reference clock provided by a GPS receiver (time) and used a separate dedicated Ethernet segment to carry the NTP traffic that is used to synchronize the routers. This separate Ethernet segment is used to have very good synchronization of the routers. Otherwise if a single Ethernet segment was used for both NTP and normal traffic this could alter the synchronization process due to the network load or the loss and delay introduced by the DCSs. In the real world a dedicated NTP server synchronized with a GPS time would be used together with dedicated NTP Ethernet segments at every location that hosts a number of routers.

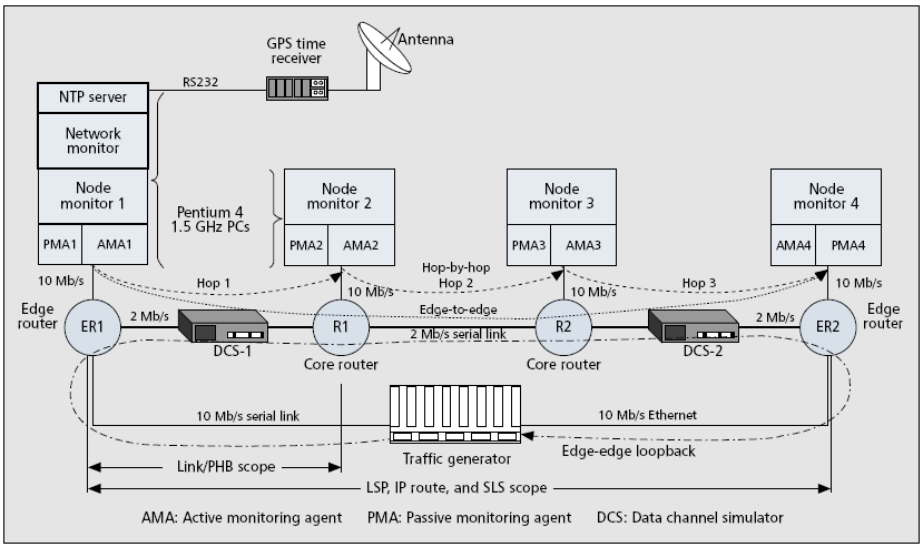


Figure 1.2 *Test-Bed Configuration [16]*

1.7 Measurement accuracy and scalability

Measurement accuracy of monitoring systems is very important aspect of network management since the operation of the network relies on the reliability and accuracy of the collected information. In a first series of tests one packet delay and loss was measured between ER1 and ER2 [10]. Delay accuracy is very good since values measured by the monitoring system were very close to the ones produced by the traffic generator. Similar behavior was observed with respect to packet loss (3.1% mean packet loss applied by DCS-2 at link 3, 3.28% mean packet loss measured. With good accuracy both for delay and packet loss it is important to also test the accuracy and scalability of the two measurement methods (per-hop, edge-to-edge) also for delay and loss. In these scenario delay and packet loss produced by the traffic generators are compared to measured aggregated values computed by NetMon using per hop measurements from NodeMons.

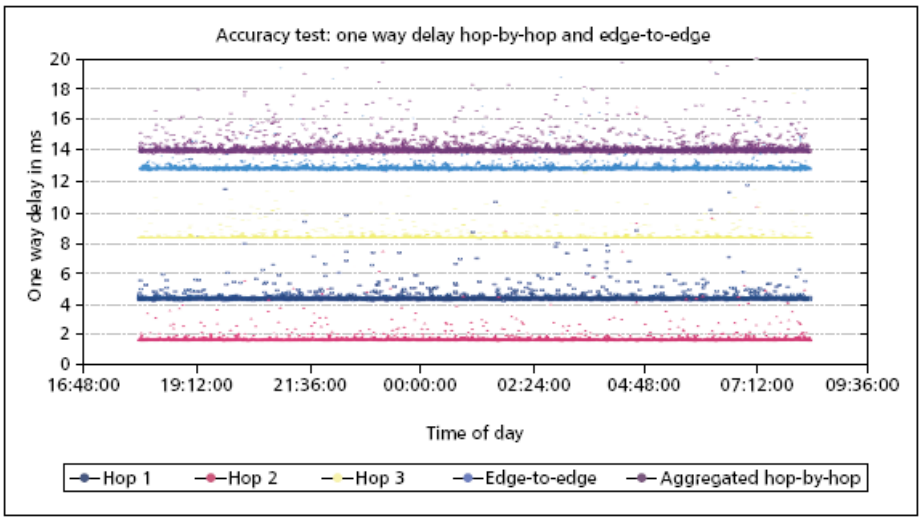


Figure 1.3 *Edge to edge and per hop one way delay [16]*

From Figure III it can be seen that the mean difference between the per hop and the edge to edge approach in computing delay is 1.1 ms. This means that there is a slight overestimation of the actual delay and as such the TE system using the per hop approach to calculate delay should be made a bit more conservative. This slight overestimation of the delay is dependent on the number of hops that are required to calculate the edge to edge result and is attributed to the fact that synthetic traffic has to cross several Ethernet segments and also due to more processing required for the per hop method. Also NodeMons have to send the per hop calculations to NetMon to do the edge to edge calculation of the result and this also introduces delay. Additionally it should be mentioned that increasing the end-to-end delays to simulate congestion by the traffic generators does not introduce larger differences between the two approaches for the same number of hops. Moreover it should be mentioned that if active monitoring agents were embedded in the routers results would have been considerably better making the per hop method very attractive to use due to the benefit that it is much more scalable. There is currently a trade off between accuracy and scalability and network administrators have to take into account that if monitoring is supported within the routers, the per-hop method is the definite choice to calculate delay.

For one way packet loss the average one way packet loss captured by the per-hop method is 5.26% and 5.16% with the edge to edge method (Figure IV). The 0.1% difference is trivial and is attributed to rounding errors. Overall, the per-hop method gave results comparable to the edge to edge method and should be favorable towards the latter due to the benefit of scalability. Similar to delay results it is observed that the only factor that affects the difference in the observed values between the two methods is the number of hops. In a similar manner if extreme accuracy is required to calculate results on very long routes for high profile traffic classes the edge to edge method should be the most preferable one. If monitoring functionality is added though in the routers, differences between the two approaches will decrease and the per-hop method should be the most preferable one.

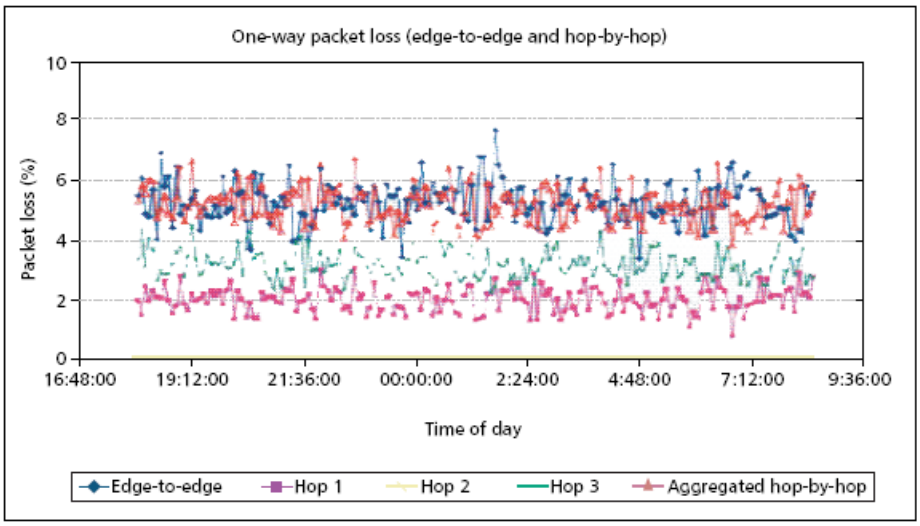


Figure 1.4 *Edge to edge and per hop one way packet loss [16]*

1.8 Measurements to support dynamic traffic engineering

To demonstrate the ability of the monitoring system to support dynamic per class of service TE, two scenarios are investigated and performance measurements are gathered when (a) multiple traffic classes on a single interface and (b) multiple traffic classes on various LSPs (interfaces) . The data that are gathered can be used to optimized network utilization and performance. These data are used by entities in the management system to dynamically alter the parameters of the network such as the bandwidth, the weights and the buffer size that is assigned to each traffic class to increase network utilization and performance [20]. Also measured data, especially in the second case of measurements can be used to take load balancing decisions in order to remap traffic that has been assigned to certain LSPs that are heavily utilized to other LSPs that are lightly utilized [21].

Using the test-bed in Figure II three traffic classes termed Expedidited Forwarding (EF), Assured Forwarding 1 (AF1) and Best Effort (BE) which represent three PHBs are implemented by LLQ along with CBWFQ. LLQ is used in the output interfaces of all routers with queues reserved for each class. With LLQ data that belong to the EF class are served before data from the AF1 and the BE class. Each class has its own bandwidth, weight, and maximum packet limit. The bandwidth assigned to a class is the throughput rate that is guaranteed to a class during congestion. EF traffic is configured to always have priority against the other queues, while AF1 and BE traffic use CBWFQ. Monitoring agents and the traffic generators are configured to support and produce traffic for all three classes. The packet inter-arrival time is configured to be exponentially distributed and the size of synthetic packets is configured to be of similar size to user traffic generated by the traffic generators. This means that CBWFQ treats synthetic and user traffic in the same way and as such the delay measured for synthetic traffic is comparable to that experienced by user traffic.

In Figure V one-way delay at hop 2 between router R1 to R2 for the scenario where one interface is used for all class in congested and non congested situations is given. In the non congested situation one-way per-hop delay for all classes is very low and similar. The link delay consists of propagation delay (1 ms), transmission delay (0.51 ms) on the links interface and queuing delay. Queuing delay is very small when the network in non-

congested. When the network becomes more congested queuing delay increases but there is a difference between the queuing delays experienced by each class.

For packet loss the monitoring agents offers us the ability to measure apart from the packets that are lost, the average number of packets in a queue and the link utilization of each of the classes (Figure VI). In the second monitoring scenario where each traffic class is bound to a different unidirectional LSP between routers ER1 and ER2, we configure the appropriate active monitors in NodeMons at the two edge routers with two jobs to monitor one-way delay and packet loss experienced by each traffic class using the LSP tunnels. The per-hop approach is not used here. To produce packet loss in order to measure it within a tunnel, the amount of BE traffic injected to the network from ER1 to ER2 is set to be more than the amount of bandwidth reserved for the queue belonging to the BE traffic class. This way we can cause an LSP to become congested. In Figure VI one-way packet losses for the three LSPs carrying different traffic is shown. Packet loss is very low for EF traffic but very high in the LSP carrying BE traffic since we inject more synthetic traffic in the BE LSP than the queue can handle. For one way delay measurement averaging occurs every 2 secs and each measurement is represented with a point in Figure VI. This timescale is quite short and proves that active monitoring can be applied

To show that the proposed monitoring system is capable of performing passive measurements as well we configure three jobs to monitor throughput on the three LSPs. Traffic generator are used to generate EF and AF1 traffic at 600 kb/s rates each and BE traffic at 800 kb/s. In figure VII the average measured throughput for the EF, AF1, and BE traffic is about 600, 600, and 700 kb/s, respectively. This is expected since BE traffic is subjected to heavy losses. The period used to measure throughput is 10 secs and its value is represented with a point in Figure VII. This timescale is quite short and proves that passive monitoring can also be applied

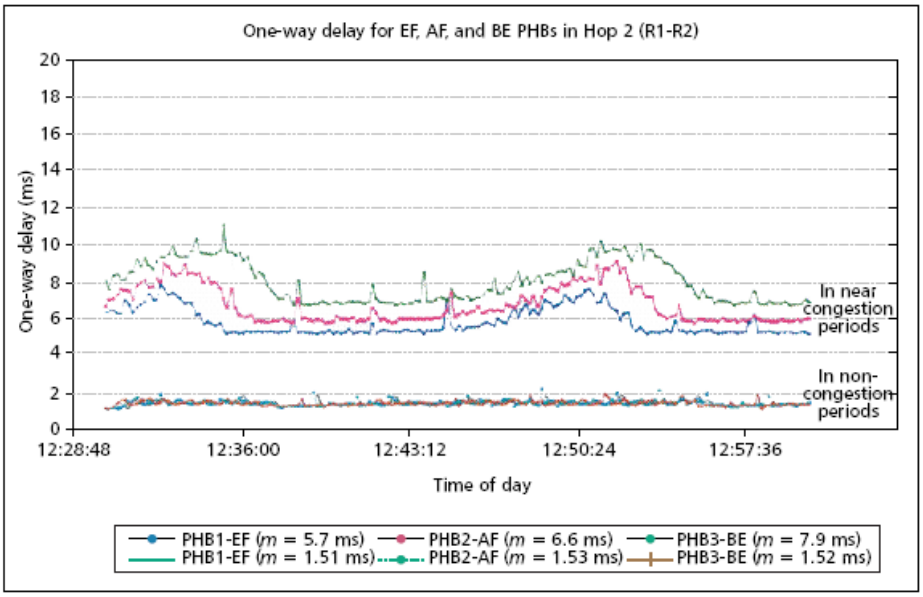


Figure 1.5 *One way delay in congested and non congestion situations for scenario where all traffic classes are bound to one interface [16]*

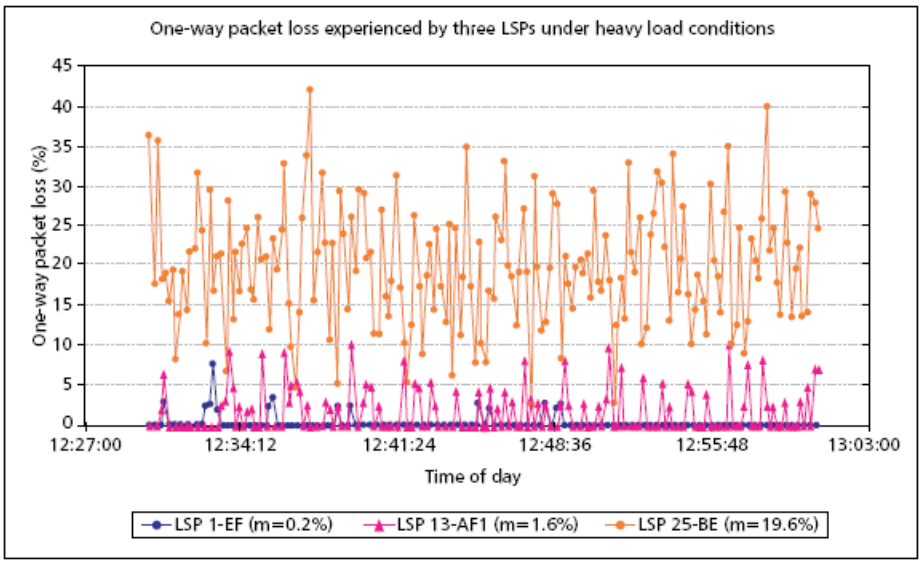


Figure 1.6 *One way packet loss for scenario where each traffic class is bound to a different LSP[16]*

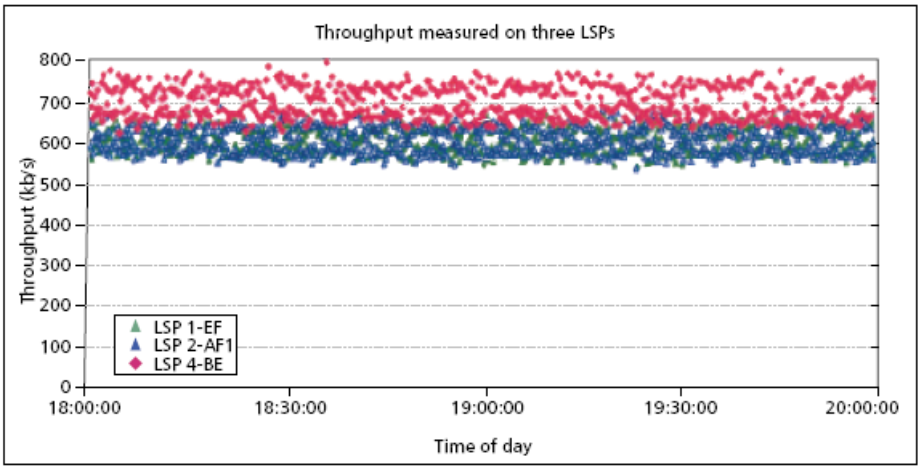


Figure 1.7 *Passive monitoring for three traffic classes for the scenario where each class is bound to a different LSP [16]*

1.9 Summary and Conclusions

Delivering QoS-based value-added IP services to customers requires careful traffic engineering of the network and its traffic because maximizing network resource utilization and at the same time meet the QoS demands of services contracted to customers can be difficult task to achieve. Traffic engineering though relies on a monitoring system that is able to capture the network's performance in order to take offline proactive and dynamic reactive measures. As such we identified the requirements of such a monitoring system for traffic engineering and then proposed effective principles to meet these requirements in order to make it scalable yet effective. To guarantee that the monitoring system has quick response times and produces minimal traffic it has to be distributed. This ensures small reaction times and helps maintain scalability as the network size increases. We applied these principles and methodologies in a monitoring system used for Traffic Engineering and assessed its

performance. Based on the experiments it is shown that the proposed monitoring system provides good accuracy in collecting data for active measurements both with an edge-to-edge method as well as with a hop-by-hop approach. We also show the ability of the proposed monitoring system to perform passive measurements. In summary it is shown that the principles that have been presented result in a more scalable monitoring system that is capable of contributing to minimizing operational traffic in order to support traffic engineered networks that can support a large number of customers.

1.10 References for section 1

[1] D. Goderis et al., "*Service Level Specification Semantics and Parameters*", Internet draft, draft-tequila-sls-02.txt, Expired Aug. 2002.
[2] S. Blake et al., "*An Architecture for Differentiated Services*", RFC 2475, Dec. 1998.
[3] For information on the various IETF working groups including DiffServ, IPPM, IPFIX, RTFM, and PSAMP, visit http://www.ietf.org
[4] Test Traffic Measurements (TTM) project of Réseaux IP Européens (RIPE) Network Coordination Centre (NCC); http://www.ripe.net/ttm/
[5] V. Paxson et al., "*An Architecture for Large-Scale Internet Measuremen*t", IEEE Commun. Mag., vol. 36, no. 8, Aug. 1998, pp. 48–54.
[6] Network Analysis Infrastructure (NAI) projects of NLANR http://mna.nlanr.net/infrastructure.html
[7] CAIDA projects; http://www.caida.org
[8] C. Fraleigh et al., "*Packet-Level Traffic Measurements from the Sprint IP Backbone*", IEEE Network, vol. 17, no. 6, Nov./Dec. 2003.
[9] IST research projects: http://www.cordis.lu/ist/; IST-INTERMON: http://www.ist-intermon.org/; IST-MoMe: http://www.ist-mome.org/; ISTSCAMPI: http://www.ist-scampi.org/
[10] A. Asgari et al., "*A Scalable Real-Time Monitoring System for Supporting Traffic Engineering*", Proc. IEEE Wksp. IP Ops. and Mgmt., Dallas, TX, Oct. 2002, pp. 202–97.
[11] P. Trimintzios et al., "*A Management and Control Architecture for Providing IP Differentiated Services in MPLS-Based Networks*", IEEE Commun. Mag., vol. 39, no. 5, May 2001.
[12] Creation and Deployment of End-User Services in premium IP networks CADENUS, http://wwwcadenus.fokus.fraunhofer.de/
[13] End to End QoS through Integrated Management of Content, Network and Terminals, http://www.enthrone.org/
[14] C. Srinivasan, Bloomberg L..P, A. Viswanathan, "*Multiprotocol Label Switching (MPLS) Label Switching Router (LSR) MIB*", Network Group RFC 3813, June 2004.
[15] T. Nadeau, C. Srinivasan, Bloomberg L.P, A. Viswanathan, "*Multiprotocol Label Switching (MPLS) Forwarding Equivalence Class To Next Hop Label Forwarding Entry (FEC-To-NHLFE) MIB*", Network Group RFC 3814, June 2004.
[16] A. Asgari, Richard Egan, P. Trimintzios and G. Pavlou "*Scalable Monitoring Support for Resource Management and Service Assurance*", IEEE Network • November/December 2004
[17] V. Paxson et al., "*Framework for IP Performance Metrics*", IETF RFC-2330, May 1998.
[18] P. Aukia et al., "*RATES: A Server for MPLS Traffic Engineering*", IEEE Network, vol. 14, no. 2, Mar./Apr. 2000, pp. 34–41.
[19] A. Asgari et al., "*Building Quality of Service Monitoring Systems for Traffic Engineering and Service Management*", J. Net. and Sys. Mgmt., vol. 11, no. 3, Dec. 2003.
[20] N. Christin and J. Liebeherr, "*A QoS Architecture for Quantitative Service Differentiation*", IEEE Commun. Mag., vol. 46, no. 6, June 2003, pp. 38–45.
[21] A. Elwalid et al., "*MATE: MPLS Adaptive Traffic Engineering*", Proc. IEEE INFOCOM 2001, Anchorage, AK, Apr. 2001.

1.11 Acknowledgements

The authors would like to thank their colleagues in the TEQUILA project especially Mr Asgari Hamid, Richard Egan and Panos Trimintzios for their big contribution to the ideas presented in this article, and the anonymous reviewers for their constructive feedback.

2 SNMP Trace Analysis

The Simple Network Management Protocol (SNMP) was introduced in the late 1980s [1] and has since then evolved to what is known today as the SNMP version 3 framework (SNMPv3) [2]. While SNMP is widely deployed, it is not clear which features are being used, how SNMP usage differs in different types of networks or organizations, which information is frequently queried, and what typical SNMP interaction patterns are in real world production networks.

There have been several publications in the recent past dealing with the performance of SNMP in general [3], the impact of SNMPv3 security [4], [5], or the relative performance of SNMP compared to Web Services [6]–[8]. While these papers are generally useful to better understand the impact of various design decisions and technologies, some of these papers lack a strong foundation because authors typically assume certain SNMP interaction patterns without having experimental evidence that the assumptions are correct. In fact, there are many speculations on how SNMP is being used in real world production networks and how it performs, but no systematic measurements have been performed and published so far.

Many authors use the ifTable of the IF-MIB [9] or the tcpConnTable of the TCP-MIB [10] as a starting point for their analysis and comparison. Despite the fact that there is no evidence that operations on these tables dominate SNMP traffic, it is unclear how these tables are read and if any optimizations are done by deployed management applications. It is also unclear what the actual traffic trade-off between periodic polling and more aperiodic data retrieval is. Furthermore, we do not generally understand how much traffic is devoted to standardized MIB objects and how much traffic deals with proprietary MIB objects and whether the operation mix differs between these object classes or between different operational environments.

This chapter describes an effort to analyze SNMP traffic traces in order to find answers to some of these questions. Section 3.1 discusses the locations from which traces have already been collected. Section 3.2 provides some initial results of our analysis; it should be noted that our research is still in progress and more detailed results will be published in a forthcoming paper. Section 3.3 discusses related work and conclusions are finally provided in Section 3.4.

2.1 Traces

Traces have been collected at several different locations. We report here analysis results covering eight traces collected at seven different locations. To easily identify the traces and the locations, we use a naming scheme essentially consisting of two numbers: The first number identifies a location while the second number identifies a specific trace collected at that location. For example, a trace name such as l01t02 refers to the second trace collected at location number 1.

Trace	description	start	hours
l01t02	national research network	2005-07-26	162.98
l01t05	national research network	2006-07-10	336.00
l02t01	university network	2006-04-21	294.62
l03t02	faculty network	2006-04-27	159.21
l04t01	server-hosting provider	2006-04-14	4.00
l05t01	regional network provider	2006-04-19	580.60
l06t01	national research network	2006-05-14	222.08
l12t01	point of presence	2006-07-10	208.02

Table 2.1 Overview of the SNMP traces

Table 2.1 provides an overview of the traces analyzed in this paper. The traces l01t02 and l01t05 have been collected at the backbone of SURFnet, which is the research network provider within the Netherlands. Note, however, that these two traces are collected with roughly a year in between and that different attachment points were used to collect the traffic. The first trace was collected near the network operation center (NOC) while the second trace was collected close to a data collection point in the network.

Trace l02t01 was collected on a university network management VLAN while trace l03t02 was collected on a faculty network. The relatively short trace l04t01 was collected at a server-hosting provider. The data was collected by capturing all SNMP traffic originating from or destined to a specific network manager.

Trace l05t01 was collected at a regional network provider network. The network utilizes many wireless point-to-point links to interconnect research institutions, government institutions and commercial organizations.

Trace l06t01 was collected on the main network management server of a national research network. Note that there are additional systems generating SNMP traffic in this network and thus the trace only describes the traffic generated by a single management system.

Finally, trace l12t01 was collected at a point of presence of another national research network. Network traffic was captured using tcpdump and stored in pcap format. In some cases, we could capture other management traffic (e.g. SYSLOG) in addition to SNMP traffic. We plan to analyze these traces in the future and relate the results to the SNMP analysis we are working on at the moment.

2.2 Analysis

The purpose of this section is to present some initial analysis results; more traces must be collected and additional analysis methods and scripts must be developed before more comprehensive conclusions on SNMP usage can be drawn. Still it is possible to present some interesting first results.

2.2.1 General Characterization

Table II provides a general characterization of the traces. The trace sizes are given in the CSV format that has been used throughout in the analysis. Most traces included all

information in the CSV file. Only traces l04t01 and l12t01 have been filtered to exclude varbind values and thus the trace files contain less information and are somewhat smaller.

Trace	size [MB]	messages	SNMPv1	SNMPv2	SNMPv3
l01t02	6369	51772136	100.0%	-	-
l01t05	14043	40072529	-	100.0%	0.0%
l02t01	77789	258010521	5.5%	94.5%	-
l03t02	130858	871361365	95.0%	5.0%	-
l04t01	10	15099	35.7%	64.3%	-
l05t01	2898	25298667	100.0%	-	-
l06t01	24683	89277889	57.4%	42.6%	-
l12t01	312	2619884	32.3%	67.7%	-

Table 2.2 General characterization of the traces

Table 2.2 shows the SNMP versions found in the traces. Despite the fact that the status of SNMPv1 and SNMPv2 is historic and only SNMPv3 is full standard, it turns out that SNMPv3 is not really used in any of our traces. Trace l01t05 contains a very few SNMPv3 messages but they do not contribute significantly to the trace; it seems that someone was experimenting with SNMPv3 while the traces were collected.

This result did not come as a surprise since earlier discussions with operators already showed that, besides for (DOCSIS) cable modem management, SNMPv3 is still not widely deployed. A second interesting observation is that some trace locations still seem to rely solely on SNMPv1, whereas the management traffic at other locations is dominated by SNMPv2.

2.2.2 Protocol Operations

Table 2.3 shows for every trace the usage of the different protocol operations. In general, the overall traffic is dominated by Get, GetNext, and GetBulk operations and their responses. Some traces do not contain any notifications and only trace l06t01 contains acknowledged notification.

Trace	Get	Next	Bulk	Set	Trap	Inform	Resp
l01t02	0.0	50.0	-	0.0	-	-	50.0
l01t05	0.0	-	50.0	-	-	-	50.0
l02t01	0.1	2.4	47.1	0.0	0.7	-	49.6
l03t02	0.3	49.8	-	0.0	0.0	-	49.9

l04t01	32.8	3.8	22.9	-	-	-	40.5
l05t01	50.0	0.0	-	-	0.0	-	50.0
l06t01	12.1	31.4	6.5	-	0.0	0.0	50.0
l12t01	1.0	49.0	-	-	0.0	-	49.9

Table 2.3 Usage of protocol operations (in %)

The traces l01t02, l02t01, and l03t02 contain a Set operations but due to the small number, they do not play a significant role in the overall traffic mix. A closer look at trace l01t02 revealed that all recorded Set operations were trying to modify the sysLocation scalar with a value of type Integer32, which obviously leads to an error response due to a type mismatch, if authentication and access control would have been successful. In trace l02t01, we observe Set requests to two proprietary MIB modules, which allow to copy configuration files to/from a device. In trace l03t02, we found that Set operations are used to trigger the download of a VLAN membership policy specification.

Table 2.3 also reveals that trace l04t01 contains significantly more requests than responses. One possible explanation could be packet loss. Upon further investigation and discussion with the network operators, we learned, however, that the day the traces were collected some systems were switched off for maintenance purposes.

trace	Get	Next	Bulk	max-reps	non-reps
l01t02	37.5%	99.3%	-	-	-
l01t05	100.0%	-	100.0%	10/50	0
l02t01	56.3%	99.9%	100.0%	1/10/20/25	0
l03t02	1.6%	99.9%	-	-	-
l04t01	100.0%	100.0%	100.0%	1000	0
l05t01	99.9%	95.6%	-	-	-
l06t01	8.7%	2.6%	0.0%	12	0
l12t01	100.0%	99.9%	-	-	-

Table 2.4 Percentage of single varbind GET, GETNEXT, and GETBULK operations and GETBULK parameters

Since the traffic is dominated by data retrieval operations, it makes sense to take a closer look how data retrieval is performed. Columns 2, 3, and 4 of **Error! Reference source not found.** show the percentage of Get, GetNext, and GetBulk requests that contain only one varbind in the varbind list. Except for trace l06t01, single varbind GetNext and GetBulk operations clearly dominate. In four traces, even the Get operations tend to be single varbind operations. In trace l06t01 we find that 86.5% of the GetBulk operations contain two varbinds and the remaining 13.5% contain eight varbinds. Furthermore, 85.8% of the GetNext operations contain two varbinds. A conclusion from this analysis is that except in trace l06t01, table retrieval is usually realized in column-by-column mode.

Columns 5 and 6 of Table 2.4 indicate the max-repetitions (max-reps) and the non-repeaters (non-reps) parameters of the GetBulk operations. The first interesting observation is that none of the GetBulk operation used non-repeaters. In traces l01t05 and l02t01, almost 100% of the GetBulk requests use 10 max-reps. The GetBulk requests in trace l04t01 always use 1000 max-reps while the requests in trace l06t01 always use the 12 max-reps.

2.2.3 Response Size Distribution

Figure 2.1 shows the cumulative distribution of the sizes of the response messages. The vast majority of the response messages are about 100 bytes long for traces that do not include the GetBulk operation. The corresponding requests are usually even smaller since they do not contain any values. As we will see later, most of the objects retrieved are actually simple numbers whose encoding is relatively compact.

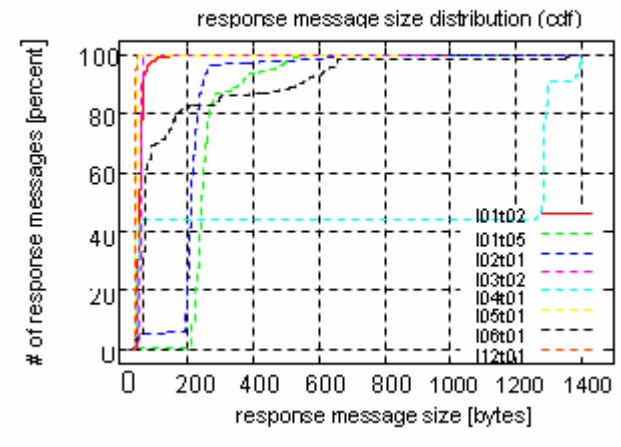


Figure 2.1 Response message size distribution (cdf)

The traces that include GetBulk requests contain larger response messages. Although almost all GetBulk response sizes could be observed, no response message was larger than roughly 1400 bytes. This implies that no fragmentation occurred on a path with an Ethernet MTU.

When comparing the four GetBulk traces, one can observe that trace l04t01 achieves a much higher percentage of large response messages. This is due to the fact that this particular management application gives the agent more freedom to generate large response messages by using a large max-repetitions parameter (1000). We have observed responses with up to 74 repetitions for l04t01. As explained above, traces l01t05 and l02t01 prefer a max-repetitions parameter of 10 and we also observe a matching number of responses with 10 varbinds. According to Figure 2, this leads to most responses being about 300 bytes long. Trace l06t01 uses the max-repetitions parameter 12 and retrieves multiple varbinds, which leads to some slightly better performance. We conclude that the setting of the max-repetitions parameter can be improved in three out of four locations and we also observe that implementations obviously do not try to dynamically adjust the max-repetitions parameter.

2.2.4 Flow Analysis

To further analyze the traces, it is useful to look at individual flows. Table 2.1 lists in columns 2 and 3 the number of CG/CR flows and the number of NO/NR flows we have

identified in the traces. The remaining columns list the number of CG/CR/NO/NR interfaces that were identified. Note that we identify interfaces and not managers or agents. Hence, it is possible that, for example, multiple CGs are running on a single multi-homed host. From discussions with operators, we know that management systems are indeed often multi-homed (and many routers are by their very nature). In particular, we know that trace l06t01 was obtained from a single management system.

Trace	cg/cr flows	no/nr flows	cg	cr	no	nr
l01t02	203	-	3	178	-	-
l01t05	8	-	2	8	-	-
l02t01	258	197	5	240	197	1
l03t02	42	20	25	20	17	2
l04t01	34	-	3	34	-	-
l05t01	117	2	9	99	2	2
l06t01	288	125	3	260	125	2
l12t01	30	6	5	26	6	1

Table 2.5 Characteristics of CG/CR and NO/NR flows

The traffic is not evenly distributed across the flows. In fact, all traces have a relatively small number of dominating flows. Figure 3 shows the number of messages per minute and the number of bytes per minute exchanged in the various flows of trace l01t02. The flows were sorted by descending number of messages per minute.

The solid curve in Figure 2.2 shows that there are some high volume flows with up to 300 messages per minutes (mpm) but also many flows with less than 20 messages per minutes. The dashed curve shows that the traffic peeks at almost 18000 bytes per minute (bpm) but goes down to less than 1000 bytes per minute for many flows. The figure also indicates that the number of messages and the number of bytes transferred are strongly correlated in this trace. This is not too surprising since we know that most requests are single varbind GetNext requests in this trace (and we will see later that most of them retrieve numbers).

The strong correlation between the number of requests and the bandwidth consumed surely cannot be expected for flows that make heavy usage of the GetBulk operation. From Figure 2.1, we already know that trace l04t01 makes efficient use of GetBulk operations by setting the max-repetitions parameter to 1000. The traffic intensity of the flows in this trace shown in Figure 2.3 proves this.

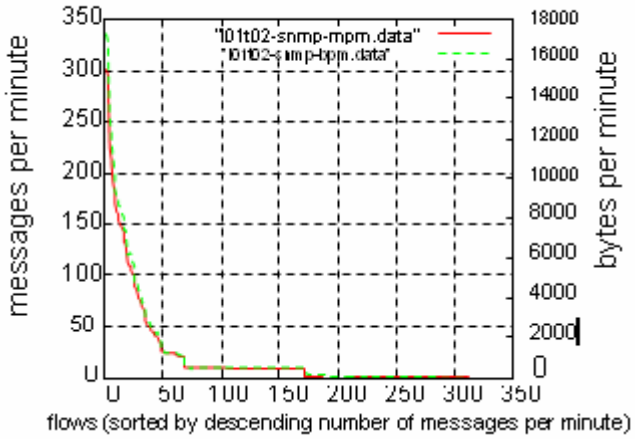


Figure 2.2 Traffic intensity of the flows in trace l01t02

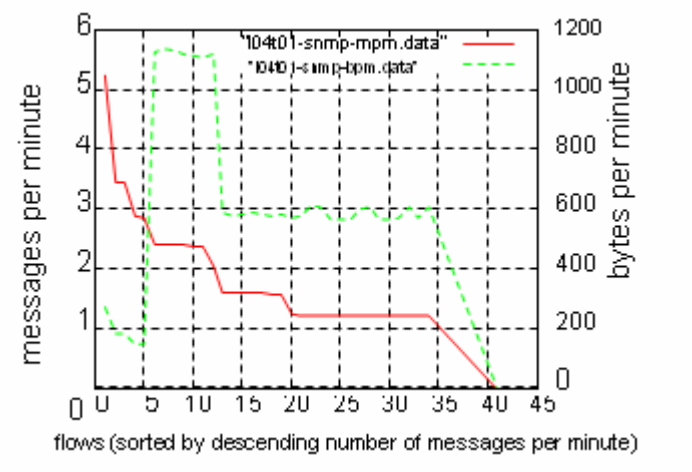


Figure 2.3 Traffic intensity of the flows in trace l04t01

The computation of the traffic flows for trace l03t02 revealed that there is one very high intensity flow, which contains 94.3% of all messages in this trace. Figure 2.4 shows the traffic intensity of the flows in logarithmic scale. This heavy flow generated close to 90,000 messages per minute. By inspecting this flow, we found that a management application got caught in a timefiltered table.

The time-filter mechanism, which was introduced in RMON2-MIB [11], can be used to retrieve only those rows of a time filtered table that have changed since a particular time. This is achieved by inserting a time-filter component in the index and excluding rows from the view that have not changed since the time-filter provided in the request. The original time-filter specification favored an implementation style where management applications that do not understand time-filtered tables could potentially run into a long laster or potentially endless loop (if the table changes faster than data can be retrieved). The update of the original time-filter definition now suggests to implement time-filtered tables in a way that avoids these problems [11].

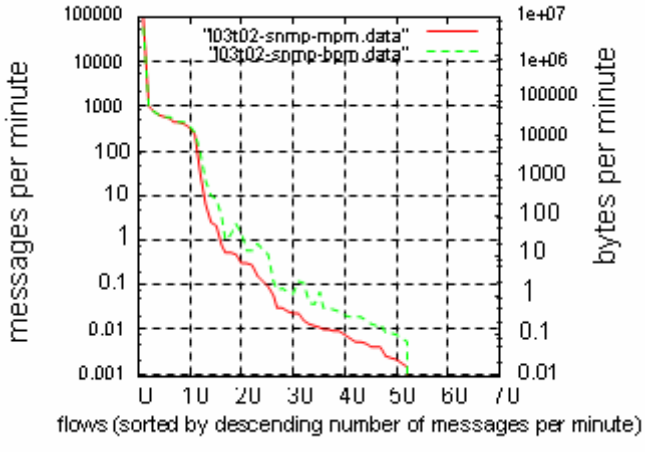


Figure 2.4 Traffic intensity of the flows in trace l03t02

2.2.5 Flow Topologies

Based on the extracted flows, it is possible to draw graphs visualizing the flow topology between CGs, CRs, NOs, and NRs. Since there are typically much more CRs than CGs, we decided to draw management systems (nodes that host CGs and/or NRs) as larger circles and managed elements (nodes that host CRs and/or NOs) as small dots. We indicate CG/CR flows with solid green arrows pointing the the CR and we use dashed red arrows for NO/NR flows. Furthermore, we indicate the relative amount of traffic at a node (as indicated by the number of messages per minute) by its gray level (black means high traffic intensity.

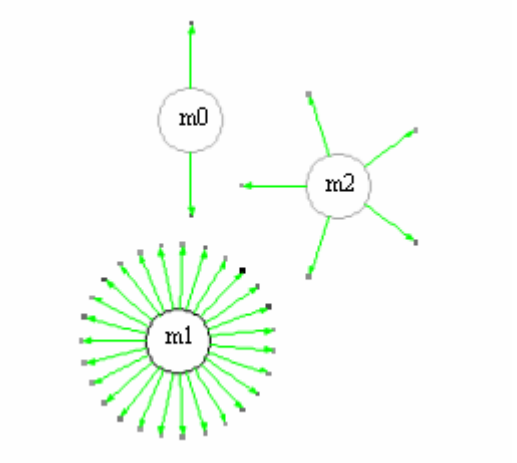


Figure 2.5 Flow topology of trace l04t01

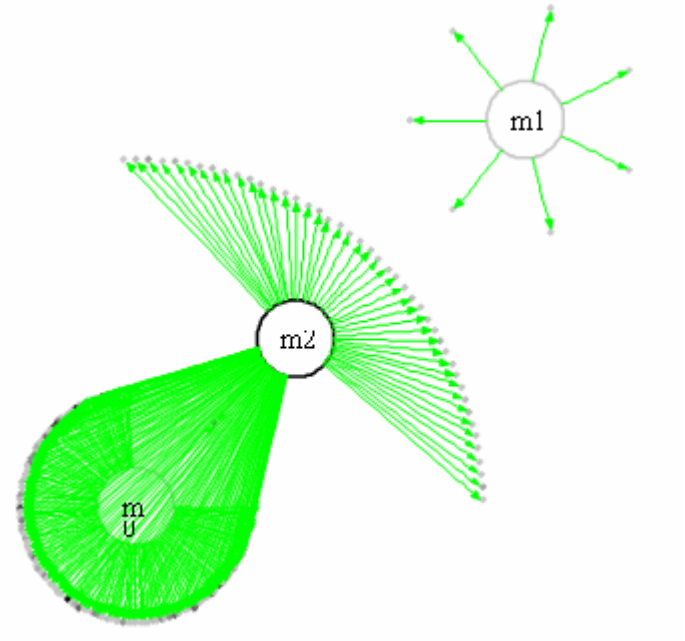


Figure 2.6 Flow topology of trace l01t02

Figure 2.5, visualizing the trace l04t01, shows a typical simple monitoring flow topology. Management interface m0 is monitoring two agents, management interface m2 is

monitoring 5 agents, and the remaining 27 agents are monitored by m1. A more complex monitoring topology is show in Figure 2.6 where we again see a rather simple low volume setup around management interface m1 plus a rather complex setup around the management interfaces m0 and m2. Surprisingly, a number of managed elements are connected to both m0 and m2.

Another interesting plot which also includes a significant number of NO/NR flows is shown in Figure 2.7. Management interface m3 acts solely as a notification receiver while management interface m1 acts as both command generator and notification receiver. The management interface m0 acts as a pure commands generator. The single dark dot and the dark coloring of m0 indicates that the traffic is dominated by a single flow between m0 and the dark managed element. As mentioned above, we know that all management interfaces in trace l01t02 belong to a single multi-homed machine.

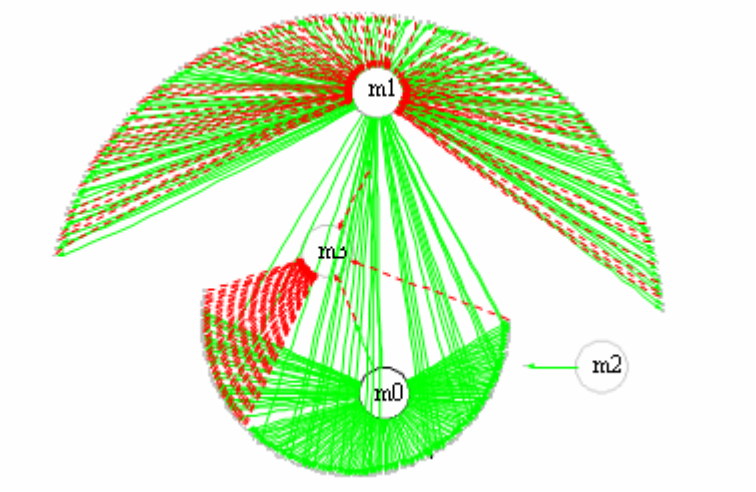


Figure 2.7 Flow topology of trace l06t01

The flow topology shown in Figure 2.8 belongs to trace l03t02 and looks kind of surprising. We see as many as 18 management interfaces talking to a single managed element interface. A closer inspection of the traces reveals that these 18 flows start consecutively, although some of them overlap. A further analysis of the flows reveals that the CRs retrieve information about printer supplies from the network element, which therefore likely is a somewhat important printer in the faculty network. We assume that the printer supply monitoring application is running on systems which get IP addresses assigned dynamically.

In Figure 2.8, we also see two notification receiver interfaces and there seem to be three nodes sending notifications to both notification receiver interfaces. The high volume time-filter traffic is exchanged between management interface m2 and the dark dot.

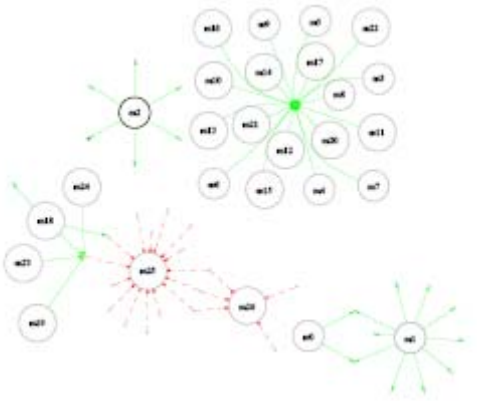


Figure 2.8 Flow topology of trace l03t02

2.2.6 Traffic Pattern

It is generally assumed that tools like MRTG [12], which fetch MIB variables at regular intervals, like every 5 minutes, generate most SNMP traffic. If such traffic were plotted as a figure that shows the average number of SNMP packets as function of the time, a straight line would be the result, provided the time interval is taken sufficiently large, for example 24 hours.

While many flows show a quite regular traffic intensity, it is important to note that this is not generally true. Figure 2.9 shows the time series for one of the highest volume flows in trace l06t01. While the regular periodic component can easily be identified, we also see significant variations. We know that the network had three major events (fibre cuts, power outages) during the measurement period. Such events clearly affect the observed monitoring traffic. It is also interesting to see that the average polling intensity seems to have changed after almost five days.

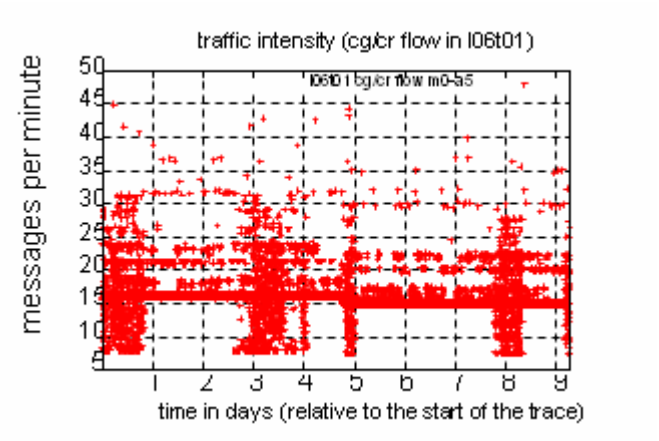


Figure 2.9 CG/CR flow traffic intensity over time l06t01

To our surprise, we found that many notification flows also carry periodic traffic, however typically at a much lower rate. Devices seem to regularly report device states (e.g., fan failures, printing engine problems) or unusual protocol states (e.g., PIM routing losses) that are not fixed. Figure 2.10 shows the traffic intensity of a notification flow

which at the beginning has a constant flow of one notification per minute. After three days, however, significantly more notifications are generated and then the flow terminates. Unfortunately, we do not have access to the notification details. All we know is that this device is an optical fibre system and that there were fibre cuts during the data collection period.

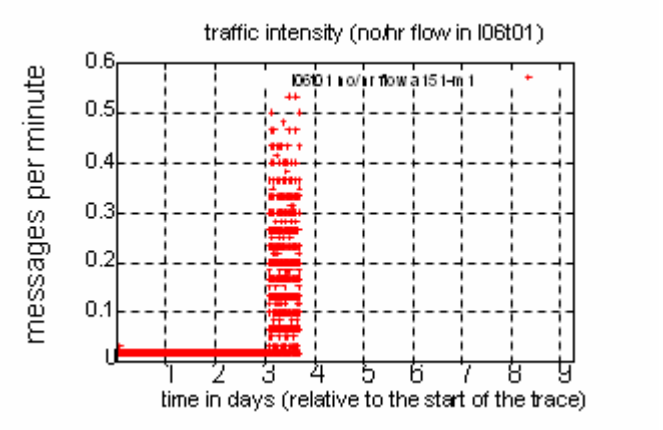


Figure 2.10 NO/NR flow traffic intensity over time l06t01

We further investigated how typical manager–agent interactions look like. Figure 2.11 shows 30 seconds of GetNext interactions for a specific CR in trace l02t01. The figure shows that the manager retrieves approximately 170 MIB objects; these objects are lexicographically ordered and represented vertically as numbers 0 to 170. The first observation is that a significant performance gain, in bandwidth as well as response time, would be possible if this GetNext sequence would be changed into a GetBulk sequence. The second observation is that the same objects are retrieved repeatedly. Further investigation has shown that many of these objects are relatively static (such as ifDescription); further performance improvements are therefore possible if more intelligent management tools would be used that cache such variables, instead of retrieving them over and over again.

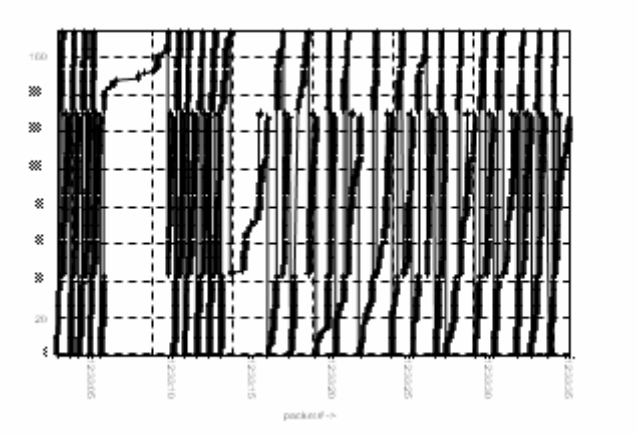


Figure 2.11 Typical GetNext sequence at l02t01

2.2.7 Data Types

Table VI shows the data types seen in response messages. The column exc lists the number of exceptions that have been seen. Response messages that include a null type are SNMP error messages (actually all noSuchName errors). However, there are also some traces where error responses contain data, especially in trace l05t01. A closer inspection revealed that this trace contains many Get requests that have values in the varbind list and it seems these requests are simply echoed back when an operation fails, without removing varbind values. We also noted that almost all requests in trace l05t01 use 0 as the request identifier—something rather dubious. We checked the other traces and found that some SNMP stacks use random request identifier while others simply increment the request identifier, which means they are predictable.

Trace	int32	uint32	uint64	oct	oid	ip	null	exc
l01t02	48.1	3.2	-	39.6	0.3	8.6	0.2	-
l01t05	13.4	21.0	52.7	12.9	0.0	0.0	-	0.0
l02t01	22.4	45.1	18.4	11.7	2.4	0.0	0.0	0.0
l03t02	2.5	95.0	-	2.4	0.1	0.0	0.0	-
l04t01	0.7	0.5	98.8	-	-	-	-	-
l05t01	2.6	80.1	-	17.0	0.0	0.0	-	-
l06t01	37.9	23.8	7.5	30.7	0.0	0.0	0.0	0.0
l12t01	48.3	51.5	0.0	0.1	0.1	0.0	0.0	-

Table 2.6 Base data type usage (in %)

Table 2.6 shows strong dominance of integral data types. This is consistent with our earlier observation that message sizes are relatively small, even when GetBulk operations with 10 or 12 max-repetitions are used. Some traces contain in addition a significant portion of octet string data. However, it seems questionable why some read-only string objects (e.g., ifDescr) are retrieved over and over again.

2.2.8 Managed Objects

The varbind names contained in response messages can be classified by matching them to MIB modules with contain the object definitions. Table 2.7 shows the percentage of varbind names that belong to the IF-MIB (IF), the BRIDGE-MIB (BR), the BGP4-MIB (BGP), the HOST-RESOURCES-MIB (HR), and the ENTITY-MIB (ENT). The last two columns aggregate names belonging to Cisco (CIS) and Centerpoint (CP) MIB modules. Note that we removed the looping timefilter flow from trace l03t02 since otherwise the RMON2-MIB would have reached 93.7%.

Trace	IF	BR	BGP	HR	ENT	CIS	CP
l01t02	40.1	-	17.6	-	10.3	30.4	-
l01t05	99.7	-	-	0.0	-	-	-
l02t01	93.5	5.5	0.0	-	0.1	0.0	-
l03t02	33.3	65.1	0.0	0.0	0.0	0.1	-
l04t01	99.7	-	-	-	-	-	-
l05t01	80.1	-	-	-	-	-	17.0
l06t01	91.3	0.0	0.0	-	0.0	2.0	-
l12t01	50.4	0.0	-	47.7	-	-	-

Table 2.7 Module usage in response messages (in %)

The IF-MIB clearly dominates in our traces. Still, the traces show differences how the IF MIB is used. The messages collected in trace l01t05, for example, retrieve regularly all 64-bit counters plus the interface names, their types, alias and discontinuity time. In trace l06t01, 32-bit counters dominate and the discontinuity indicator is ignored. Trace l02t01 shows that all columns of the ifTable and the ifXTable are retrieved regularly (including deprecated and index columns). A more precise analysis on how data retrieval takes place and how to infer the data retrieval logic of management applications is beyond the scope of this paper.

2.2.9 Notifications

Five traces contain Trap or Inform messages carrying event notifications. In trace l02t01, about 52.1% of the notifications are fan failure notifications that are repeated periodically. Some 42.2% are interface up/down notifications while the remaining notifications are HP and Avaya vendor specific. In trace l03t02, we found that all notifications were reporting printer problems. Trace l05t01 contains only Cisco notifications related to TCP session teardowns and configuration changes.

Trace l06t01 has 26.1% BGP and 8.1% PIM routing related notifications. Some 20.0% are sensor threshold crossing notifications while 13.2% are Cisco notifications related to TCP session teardowns. Note, however, that 21.3% of the notifications in this trace could not be analyzed due to a data conversion error. Trace l12t01 contains 68.5% authentication failure, 14.8% cold start and 16.7% shut down notifications.

2.3 Related work

Several recent papers discuss the performance of SNMP [3]–[5], [7], [8], [13]. This work is complementary as it aims at providing empirical data about the usage of SNMP in production network. This data is needed to select realistic scenarios and models for evaluating SNMP performance. Perhaps some of the papers mentioned above need to be revisited once we better understand how SNMP is used in production networks.

A static (compile time) analysis of MIB module definitions is reported in [14]. One conclusion was that MIB modules contain a large number of integral data types. So far, our traces also show a strong usage of integral types. The analysis in [14] also showed that advanced router vendors prefer these days to define 64-bit counters and it seems

that 64-bit counters are also preferred in some of the traces. The compiler backend described in [14] makes certain assumptions about the behavior of SNMP implementations, for example to predict likely encoding sizes. The work reported in this paper helps to provide a basis for these assumptions.

2.4 Conclusions and future work

After more than fifteen years of operational experience with SNMP, it is important to capture and analyze SNMP traffic traces to learn how SNMP is used in practice. Such knowledge is valuable for IETF protocol and MIB designers to understand which protocol features are used (or not) in practice, equipment vendors for optimizing their SNMP implementations, tool developers to identify potential improvements in their tools as well as researchers who compare new management technologies to that of SNMP or analyze modifications of SNMP.

This paper presents an approach to capture and analyze such traces. As part of our research, we have developed some analysis tools, which are openly available, and we collected traces from several different locations. Our results show that SNMP is primarily used for monitoring, and not for configuration purposes. Despite the fact that the IETF declared SNMPv1 and SNMPv2 as historic and SNMPv3 as standard, we were unable to capture many SNMPv3 packets. Some locations still rely on SNMPv1, whereas others have moved to SNMPv2 to take advantage of the GetBulk operation to improve performance and 64-bit data types. Although in many cases the flow of SNMP data is relatively constant, there are also locations where SNMP is used in some very short bursts. We also observed that a significant portion of SNMP walks use a single varbind element and that discontinuity objects are seldom used to deal with counter discontinuities. We found that the IF-MIB is by far the most popular MIB module and that integral data types are heavily used.

This paper describes work in progress. More research is needed to develop statistically sound traffic models and investigate, for example, errors, use of advanced protocol options, retrieval of outdated MIB objects and the way current tools implement table retrieval. Preliminary results indicate that significant performance improvements are possible.

2.5 Reference for section 2

[1] V. Cerf, "IAB *Recommendations for the Development of Internet Network Management Standards*", Network Information Center, SRI International, RFC *1052, Apr. 1988.*

[2] D. Harrington, R. Presuhn, and B. Wijnen, *"An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks*", Enterasys Networks, BMC Software, Lucent Technologies, RFC 3411, Dec. 2002.

[3] C. Pattinson, *"A study of the behaviour of the simple network management protocol"*, in Proc. 12th IFIP/IEEE Workshop on Distributed Systems: Operations and Management, Nancy, Oct. 2001.

[4] X. Du, M. Shayman, and M. Rozenblit, "*Implementation and Performance Analysis of SNMP on a TLS/TCP Base*", in Proc. 7th IFIP/IEEE International Symposium on Integrated Network Management, Seattle, May 2001, pp. 453–466.

[5] A. Corrente and L. Tura, "*Security Performance Analysis of SNMPv3 with Respect to SNMPv2c*", in Proc. 2004 IEEE/IFIP Network Operations and Management Symposium, Seoul, Apr. 2004, pp. 729–742.

[6] T. Drevers, R. van de Meent, and A. Pras, "*Prototyping Web Services based Network Monitoring*", in Proc. 10th Open European Summer School (EUNICE 2004) and IFIP WG 6.3 Workshop, Tampere, June 2004, pp. 135–142.

[7] A. Pras, T. Drevers, R. van de Meent, and D. Quartel, "*Comparing the Performance of SNMP and Web Services based Management*", IEEE electronic Transactions on Network and Service Management, vol. 1, no. 2, Nov. 2004.

[8] G. Pavlou, P. Flegkas, S. Gouveris, and A. Liotta, "*On Management Technologies and the Potential of Web Services*", IEEE Communications Magazine, vol. 42, no. 7, pp. 58–66, July 2004.

[9] K. McCloghrie and F. Kastenholz, "*The Interfaces Group MIB*", Cisco Systems, Argon Networks, RFC 2863, June 2000.

[10] R. Raghunarayan, "Management Information Base for the Transmission Control Protocol (TCP)," Cisco Systems, RFC 4022, Mar. 2005.

[11] S. Waldbusser, "*Remote Network Monitoring Management Information Base Version 2*", RFC 4502, May 2006.

[12] T. Oetiker, "*MRTG - Multi Router Traffic Grapher*", in Proc. 12th Conference on Large Installation System Administration (LISA XII), Boston, Dec. 1998.
[13] V. Marinov and J. Schönwälder, "*Performance Analysis of SNMP over SSH*", in Proc. DSOM 2006. Dublin: Springer LNCS 4269, Oct. 2006, pp. 25–36.
[14] J. Schönwälder, "*Characterization of SNMP MIB Modules*", in Proc. 9th IFIP/IEEE International Symposium on Integrated Network Management. IEEE, May 2005, pp. 615–628.

3 Practical SNMP usage patterns on an operational network

Complex high-performance networks require proper management techniques and software to provide users with appropriately high reliability and overall service quality. As networks grow from local area coverage with relatively low number of devices to WANs, management becomes even more complicated and makes network operators face new challenges in various areas related to network expansion, such as scalability, cost and general resource consumption. It is important to properly evaluate management solutions in use to check whether they meet the requirements and expectations of operators and users. Selecting appropriate management model and particular management technologies (protocols, whole frameworks etc.) is not a trivial task, therefore analysis of performance of management solutions used in existing networks should be performed – to provide readers with an overview of effectiveness of a running, live management system. Such analysis was conducted by Poznan Supercomputing and Networking Center as the part of EMANICS Network of Excellence research on management of next generation networks..

3.1 SNMP usage at PSNC

Poznan Supercomputing and Networking Center is responsible for operation and maintenance of two large networks (apart from its internal local area network) – PIONIER National Research and Education Network and POZMAN metropolitan network. Both of those networks are managed in a centralized way – devices are monitored and configured remotely by the PSNC Network Operation Center located in Poznan. Nodes are controlled by management stations (separate one for each network operated by PSNC) running specific network management software (such as HP OpenView Network Node Manager module, FSC3000 Element Manager, PSNC-developed Lambda Monitor or Foundry Networks IronView Network Manager). Dominant protocol in use is Simple Network Management Protocol (SNMP), mainly due to its commonness (which results in easy interoperability between solutions offered by various vendors and support and maintenance costs) and relative simpleness (low number of protocol's own messages lowers the cost of devices supporting it). These properties are important factor in case of management of large networks with a high growth potential. It is essential to take system scalability into consideration when planning expansion of such network – which is determined by performance of network management protocols used. The following article covers the subject of SNMP performance, basing on analysis of SNMP traffic traces collected from network management systems working in PIONIER NREN. Traffic traces embrace 1 month period of time (between 14.08.2006 and 15.09.2006). SNMP data was captured using a Linux-based workstation set up to capture all the management traffic coming to / from the central monitoring station on PIONIER network. This way a complete picture of management traffic was obtained – such a solution was possible due to centralized management being implemented at PSNC.

3.2 Statistics and trace analysis

Basic statistics were computed using Perl scripts delivered with SNMPDUMP package (developed by IUB and widely used throughout the EMANICS project activities). The approximately 31-day long SNMP trace was analyzed to compile some basic statistics about messages exchanged in the management process.

One of the most descriptive statistics of the system managed using SNMP protocol is the breakdown of messages by protocol version. For example, SNMP version used indicates the security model throughout the network, which then indicates roughly how resource-hungry devices and / or management software may be. Cryptography always means increased computational power requirements, the encryption process itself is CPU-hungry and the security model defined within SNMPv3 specification requires additional user and key management infrastructure for its USM (User-based Security Model) to provide increased reliability and independence from availability of other authentication methods. Taking all this into consideration, SNMPv3 is not as commonly deployed as earlier protocol versions.

Table 3.1 shows the breakdown of SNMP protocol versions and messages within the PIONIER network.

OPERATION	msg count	SNMPv1	msg count	SNMPv2c	msg count	SNMPv3	msg count	TOTAL
get-request:	3080645	14.0%	5742	0.0%	2	0.0%	3086389	14.0%
get-next-request:	7024583	31.9%	11785	0.1%	0	0.0%	7036368	32.0%
get-bulk-request:	0	0.0%	751	0.0%	0	0.0%	751	0.0%
set-request:	0	0.0%	0	0.0%	0	0.0%	0	0.0%
trap:	1721607	7.8%	0	0.0%	0	0.0%	1721607	7.8%
trap2:	0	0.0%	93208	0.4%	0	0.0%	93208	0.4%
inform:	0	0.0%	0	0.0%	0	0.0%	0	0.0%
response:	10048676	45.7%	4919	0.0%	0	0.0%	10053595	45.7%
report:	0	0.0%	0	0.0%	5	0.0%	5	0.0%
unidentified:	0	0.0%	0	0.0%	9	0.0%	9	0.0%
summary:	21875530	99.5%	116405	0.5%	16	0.0%	21991951	100.0%

Table 3.1 Message and protocol version structure within PIONIER NREN

It is clearly visible that SNMPv1 is the dominant protocol in use in PIONIER network. This is mostly because of the simplicity of this protocol, as well as its compatibility with wide range of hardware and software. Some SNMPv2 features are in use as well, although to a much lower extent than original SNMPv1 ones. Specifically, *trap2* messages are the dominant use of SNMPv2 protocol, yet still version 1 traps are far more common on the examined network, as shown in Table 3.2. The reason why there is virtually no SNMPv3 traffic within PIONIER NREN is that HP OpenView Network Node Manager in use at PSNC does not support this protocol version. Another monitoring station, supporting SNMPv3, is planned to be deployed to allow SNMPv3 monitoring of PIONIER network devices.

NOTIFICATION	NUMBER	% of all traps
trap:	1721607	94.9%
trap2:	93208	5.1%

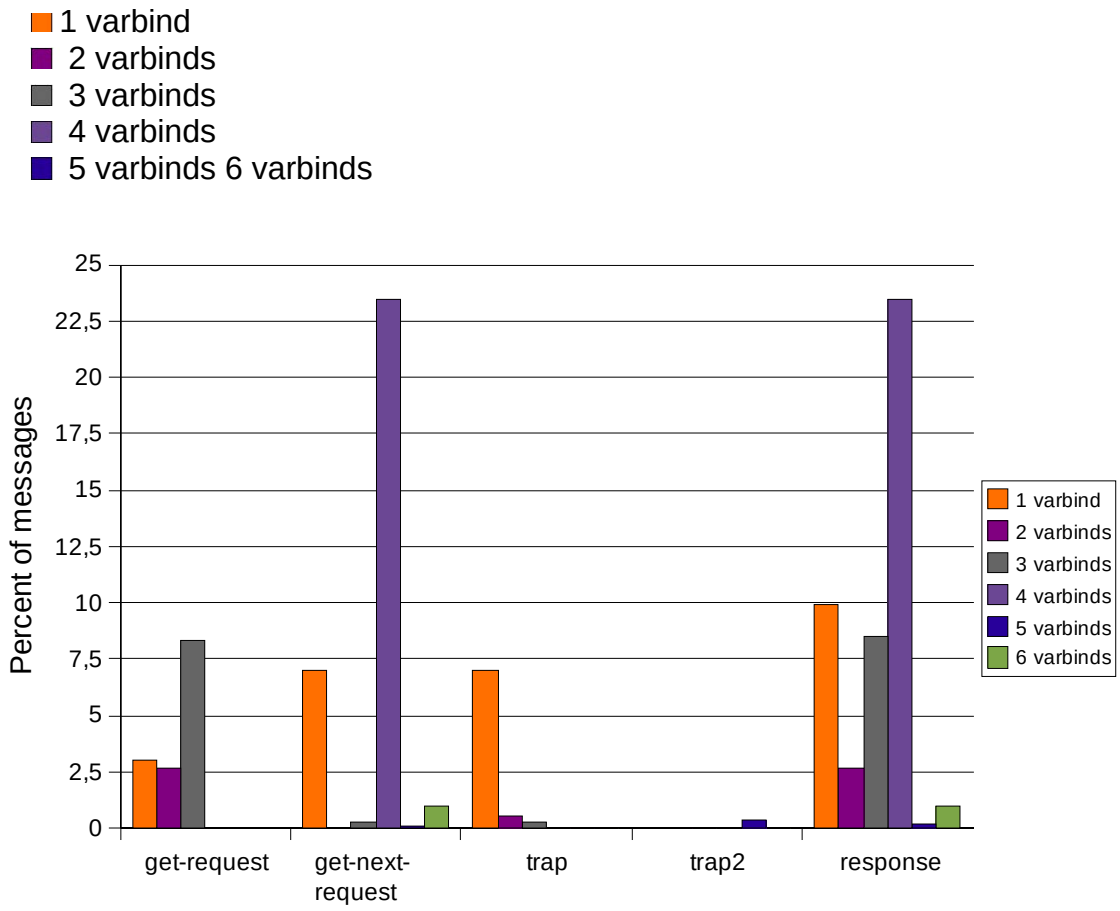
Table 3.2 Trap versions

Table 3.3 shows that there were also some *get-bulk* requests (these are a part of SNMPv2 specification), but overall *get* requests remain dominant. This of course may result in less efficient bandwidth usage, as protocol overhead related to the use of multiple *get* or *get-next* requests is obviously higher than the one related to the use of *get-bulk*. The high number of *get-next* requests shows that MIB tree was more often browsed through sequentially than randomly accessed. Overall, the use of *get-bulk* requests was quite rare and the MAX-REP parameter, defining the maximum number variables other than "non-repeaters" (defined by NON-REP parameter), is quite low (most of *get-bulk* requests contain 5 or 6 "repeaters" with MAX-REP set to 5), meaning that not much bandwidth was saved by using this SNMPv2 feature.

OPERATION	NON-REP	MAX-REP	VARBINDS	NUMBER	%
getbulk:	0	5	12	20	2,7%
getbulk:	1	5	7	314	41,8%
getbulk:	0	5	5	386	51,4%
getbulk:	0	5	6	30	4,0%
getbulk:	0	5	4	1	0,1%

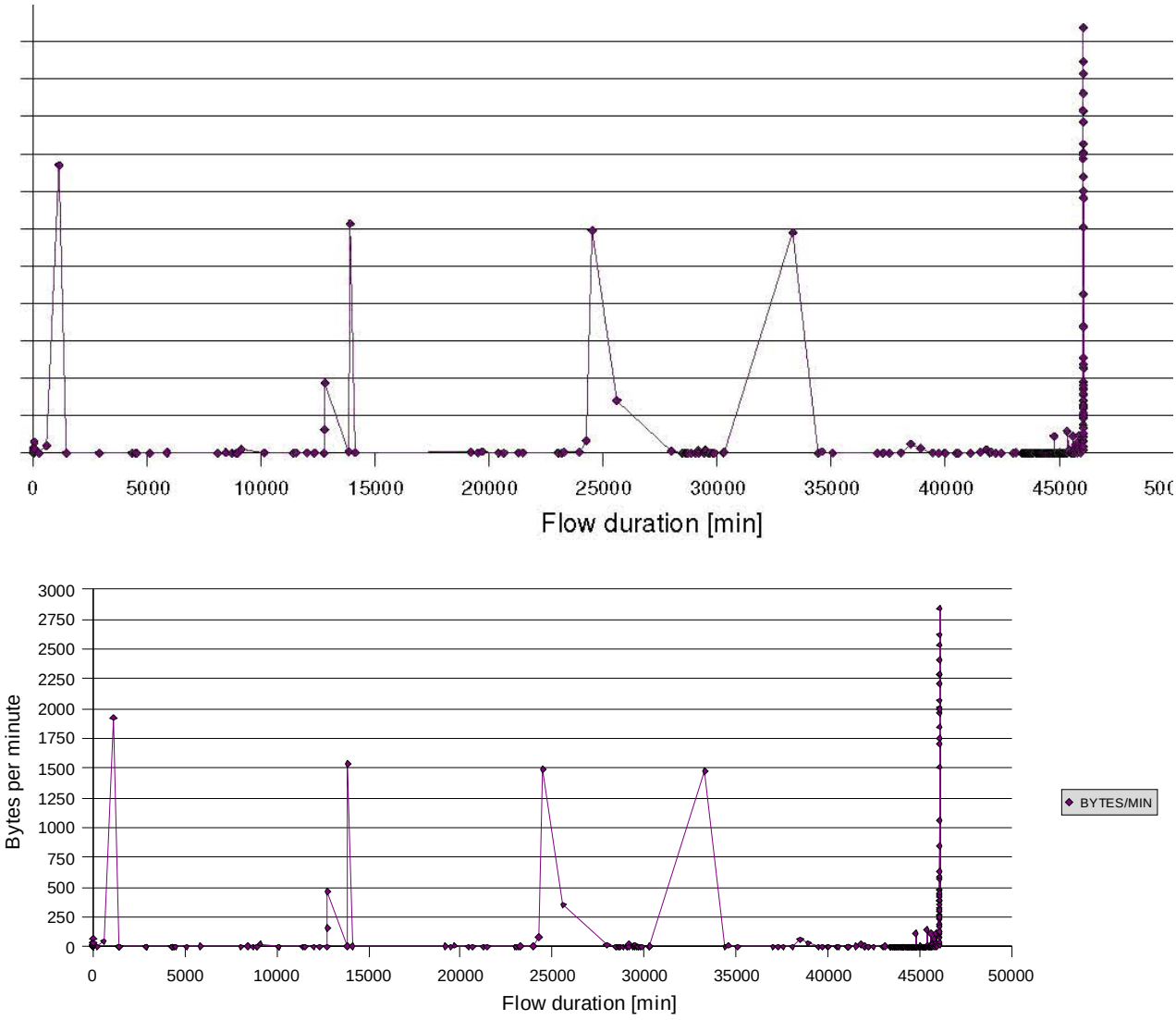
Table 3.3 Get-bulk requests broken down by MAX-REP and varbinds number

Taking a look at another statistics related to protocol overhead, one should pay attention to number of varbinds transmitted per PDU distribution. This is illustrated by Graph 3.1 (with additional information about g*et-bulk* requests in Table 3.3, as total number of *get-bulk* requests is too low to include this data on Graph 3.1). It is obvious that the more varbinds are encapsulated in single SNMP message, the lower the overhead is because of reduced header redundancy. The most common number of varbinds related to single message is 4 – and the number is connected to *get-next* requests, meaning that most possible tables inside examined MIBs were read by groups of 4 values. The amount of single varbind PDUs is fairly high as well, though not dominant. A significant amount of those messages are traps sent from agent to supervisor. Taking into consideration that traps are often alarms triggered by value of some parameters being out of defined boundaries, this is quite normal behaviour and should not be regarded as a potential performance threat. 3 varbinds per message is also a common scenario – it is nearly entirely related to *get* requests, as the impact of *get-next* requests and *traps* is negligible in this case.



Graph 3.1: Percentage of PDUs with given varbinds number

Another statistic enriching our knowledge of the way the PIONIER network is being maintained is the distribution of traffic over time. By comparing the amount of traffic generated by short-term flows (these may be treated like burst in the time scale of the whole SNMP trace) with the amount of traffic generated by long term packet flows it is possible to tell whether such “bursts” influence overall system performance in a significant way. Such comparison is illustrated by Graph 3.2.

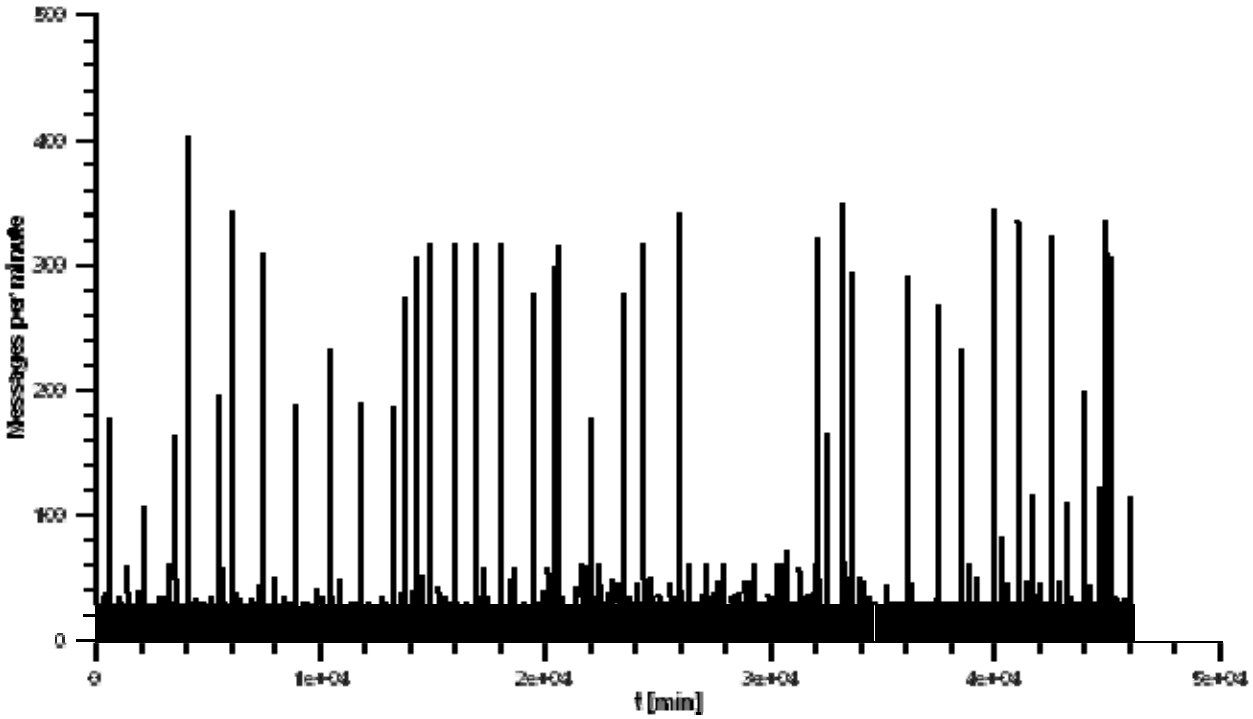


Graph 3.2 Amount of traffic related to flows of different length

As expected, most of the management traffic is related to flows that exist during the whole time the SNMP trace describes. Several traffic flows, however, existed for a shorter period within the trace timescale. There were only a few such flows identified during the whole month period of captured traffic. Some of them had a lifetime of only few minutes (or few hours) – this may be the result of some configuration changes taking place within the network or increased device polling frequency after receiving notification about some event that requires increased attention (as defined by "trap-directed polling" approach). Taking into consideration the number of flows lasting for the whole month with a substantial bandwidth usage, these short-term traffic values are relatively insignificant when talking about general system performance, though they need to be taken into consideration while estimating possible peak management traffic values.

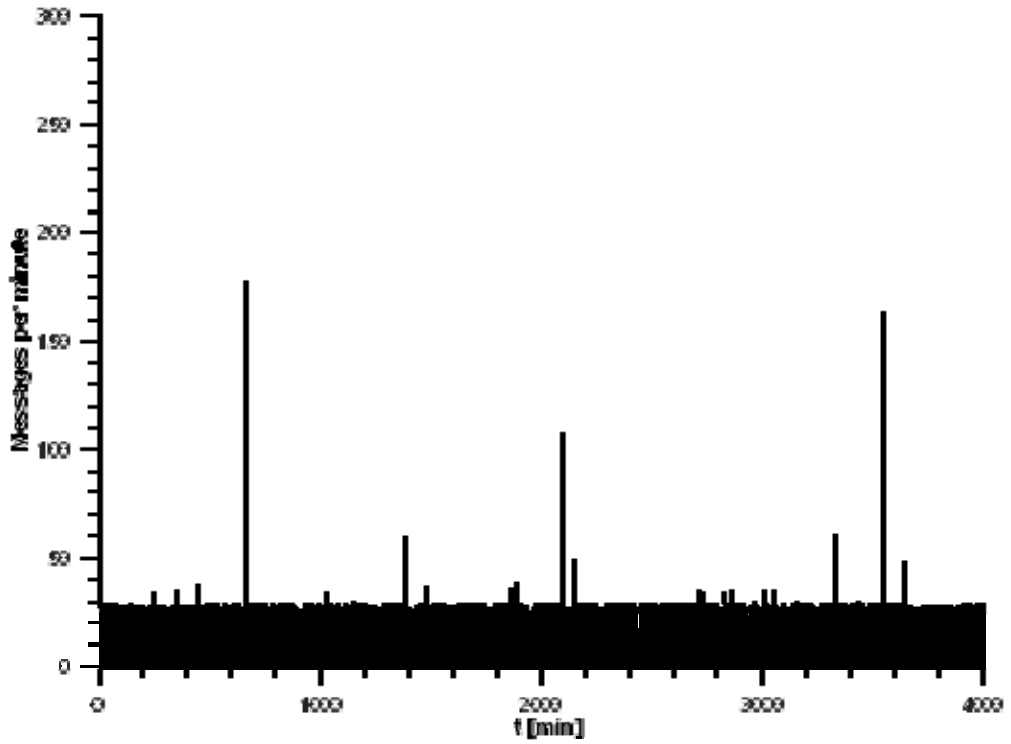
Other flows that consume a lot of bandwidth (and last much longer) may be a result of adding new devices (nodes) or parameters to monitored information list. Precise reasons for this behaviour could be identified by finding points of flow start and end on the trace timeline. This is not a part of this analysis, though. We may, however, try to understand the temporal traffic distribution better by analyzing information about the distribution of messages over time. Graph 3.3 and Graph 3.5 picture the distribution of

SNMP traffic within the captured trace timeline by providing information about number of messages exchanged each minute and number of bytes exchanged each minute.



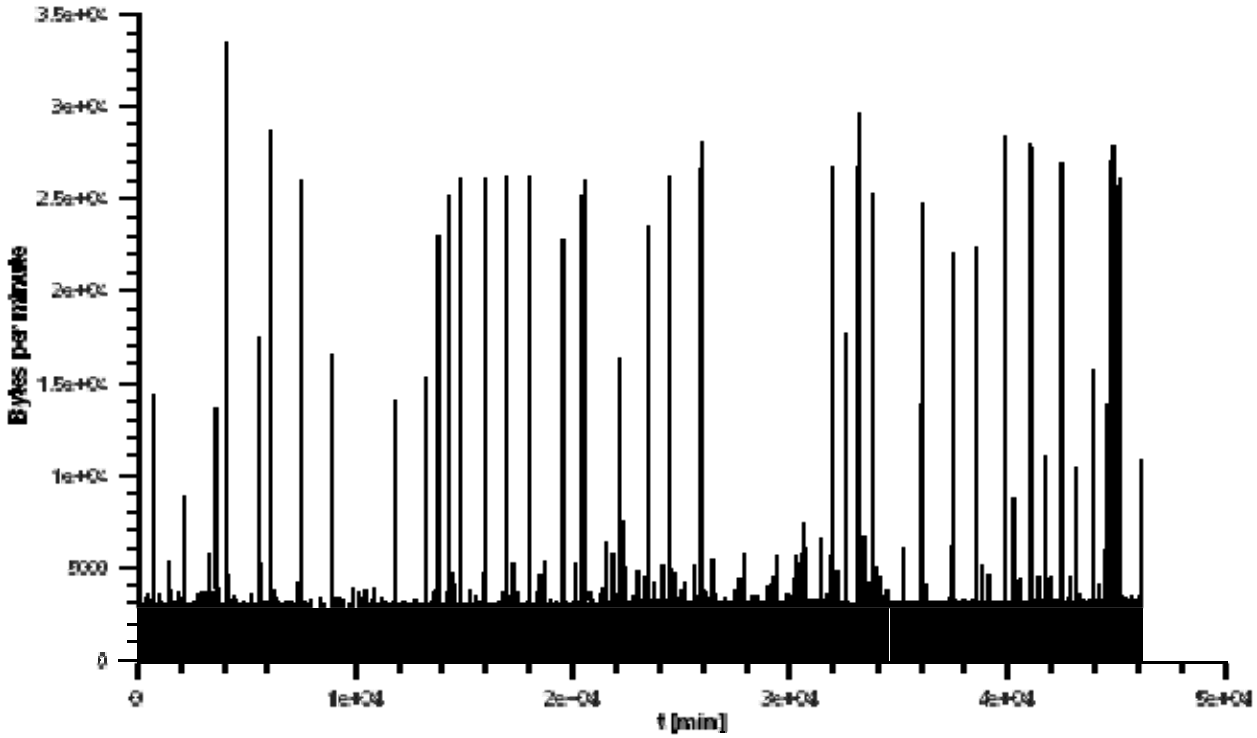
Graph 3.3: Message count per minute during the whole capture period.

Graph 3.3 shows that there is a nearly constant traffic of 30 messages per minute, with occasional bursts causing the average number of messages to go up to value of 400 per minute. These bursts seem to occur with a stable frequency – with the interval of about 1500 minutes, that makes roughly 24 hours. After looking into this issue, we found out that this is due to scheduled refresh of SNMP monitoring station database, which is performed roughly once per day. This kind of scheduled activity can be confirmed by looking closely at a 3-day period of captured traffic (see Graph 3.4). It is clearly visible that 3 most significant peaks appear every 24 hours, though peak value is significantly different between the second peak and the other two, which may point that some other set of operations takes place at the same time as the daily scheduled ones are run. Nevertheless, traffic bursts appear periodically and should be taken into consideration when assessing bandwidth requirements.



Graph 3.4 Message count per minute within first 4000 minutes of trace

Graph 3.5 looks very similar to graph Graph 3.1, as it presents a very similar aspect of SNMP traffic analysis. By comparing those two pictures, a conclusion can be made that the number of messages exchanged defines bandwidth requirements, size of messages exchanged being less significant factor.



Graph 3.5 Bytes per minute during whole trace capture period

It is also worth mentioning that the relatively short period of complete inactivity at about t=34000 minutes (see Graph 3.3 and Graph 3.5) is the result of loss of power that occurred during data collection and the subsequent configuration loss on the network switch the trace collecting machine was connected to. This, however, does not appear to influence overall statistics significantly and in fact it makes the analysis more trustworthy, as infrastructure failures are not unusual in the real world.

3.3 Summary

Overall, the PIONIER network does not benefit much from extended SNMPv2 / v3 capabilities. Most of the management traffic captured was SNMPv1, with small amount of SNMPv2c packets and only few SNMPv3 messages captured. Yet if an upgrade was considered, it would not be because of inadequate performance of the management system. Network management traffic statistics show clearly that SNMPv1 does its job well enough even in case of such big network as the country-wide PIONIER NREN. Tool included with SNMPDUMP package identified 240 agents by analyzing SNMP traffic trace data. Each of them is being polled for the purpose of statistics gathering and verification of multiple operational parameters. Still, resulting traffic is at only about 9,83 kbit/s.

The main reason for migration towards SNMPv3 would be improved security, as SNMPv1 and community-based SNMPv2c do not offer any acceptable security model. Yet SNMPv3 USM is considered costly and complicated and as such it does not appear to be as widespread as earlier SNMP protocol versions, even though SNMPv1 and SNMPv2 are currently officially considered “historical”. Work has been done by the community to introduce SSH Security Model to SNMP to lower deployment costs and

encourage wider acceptance of this protocol. This is an interesting proposal that is worth considering when choosing enhanced security management solution.

To summarize the analysis in one sentence, we might say that for now SNMPv1 seems to perform well in vPIONIER network, although SNMPv3 capable monitoring station is planned to be introduced to PIONIER network devices monitoring.

4 JMX Benchmarking: improved data analysis

4.1 Objective

INRIA has previously conducted some performance tests (benchmarking) on some aspects of the Java Management eXtension (JMX) framework. These tests are purely synthetic and target one single JMX agent (MbeanServer). A first simple analysis based on a couple of plots and comparisons have been published [gres05] (in french).

The idea of this work it to redo analysis using advance statistical and data mining methods in order to:

- find new correlation between test factors,
- to check previous results.

4.2 Experimental testbed

4.2.1 JMX background

JMX is a client-server (manager-agent) oriented management java middleware. The agent, the *MBeanServer* is in charge to expose real managed resources. These resources are indeed mapped on objects: the *MBeans* contained into the agent. One *MBean* features a X500/ldap like name, some attributes and some operations. Attributes can be remotely accessed from a manager by various means (HTTP, JMX remote protocol over RMI, SOAP). There is no direct binding between manager and MBeans so any call goes through the *MBeanServer* interface and fully qualified named of the object must be provided each time.

At run time one *MBean* is an instance of a dedicated class. There are several way to get this instance: i- code a static class and interface following a precise inheritance and naming scheme: the *standard MBean* and instanciate ; ii- code a class which just inherits from a dedicated class: the *dynamic MBeam* and instanciate ; iii- use a kind of flexible factory to directly get instances: the *Model MBean*.

4.2.2 Injectors, system under test and measurements

The system under test is the agent and only the acces to attributes are tested (service under test). This access is done by remotly invoking the *getAttribute* method on the *MbenServer* interface. So the agent is sollicitated by one or several dedicated managers: the “injectors” which regulaty call the agent’s *getAttribute* method. The agent is not linked to any actual resource (application, device...) so **only** the JMX layer overhead is under test. The agent can easily be generated for a set of test factors (kind of remote protocols, number of *MBean*s...).

The injector locally logs over time each successful *getAttribute* calls with its applicative round-trip delay. On the agent node the SAR/systat[1] tool is use to collected utilization metrics (cpu, network, memory) overt time.

Figure 4.1sumarizes the experimental set up which generated the raw data.

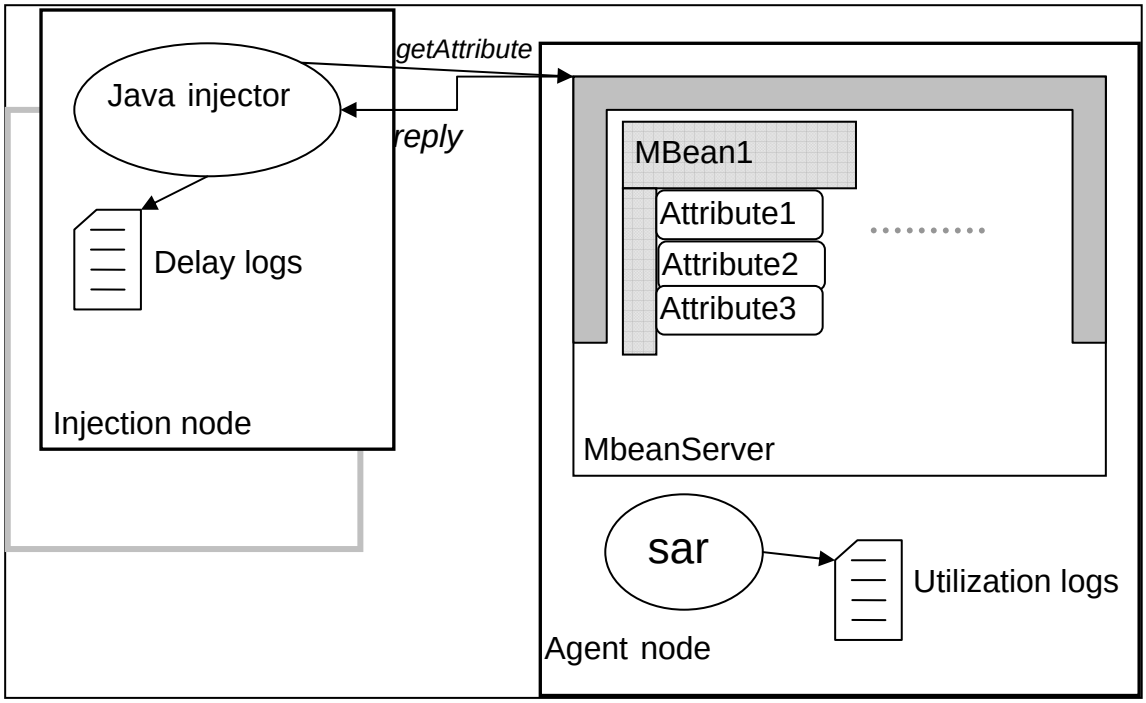


Figure 4.1 overall test architecture

4.3 Raw data candidate for analysis

Using the previously described test architecture a large serie of measurements have been conducted. For one test run the following **test factors** were settable (a mnemonic name is given in `courrier` font for each of them and their possible values. These mnemonics will be used in the statistic analysis sections):

- The vendor of the JMX implementation under test: *sun* reference implementation or *MX4J*[2]. (*TypeJMX*)
- The kind of *MBeans* (`typembean`) which are populating the MbeanServer: *standard* (`std`), *dynamic* (`dyn`) or *model* (`mod`).
- The number of *Mbeans* instanciated within the agent.
- The number of attributes exposed by any *Mbean* (`nbattr`).
- The kind of remote connection (`typeconnection`) between manager and agent: standard jmx remote connection over rmi (`rmi`) or soap adaptor (`soap`).
- The number of *get* requests injected per second at the agent interface.

[1] See: http://perso.orange.fr/sebastien.godard/faq.html
[2] See: http://mx4j.sourceforge.net

For each run, the average, standard deviation, maximum, minimum of the following metrics are calculated from the log files (some mnemonic name for each case of each metric are given in time font):

- Number of correctly served *get* request per second (tmin)
- Percentage of used cpu (cpumean, cpumin, cpumax)
- Percentage of used memory (memmean, memmin, memmax)
- Network bandwidth (in Kbytes/s)

As a result a data table of 22 columns and 382 rows (382 test runs corresponding to about 130 test hours) has been delivered by INRIA to UPI.

4.4 Statistical analysis of the performed experiments

Methods and tools

We used the standard TT test for a first analysis. The research question was formulated in hypothesis testing framework.

The null hypothesis: H_0: the means are equal for a metric

The alternative hypothesis H_1: the means for a metric are unequal for a given factor value.

The significance level α = 0.05 ; The computed value of the statistic on the specific data in the study is compared to a critical value that takes into account the degrees of freedom usually expressed in terms of the sample size. Basically, the *significance level* (probability level), usually denoted by *p*, is taken from statistical tables. The significance level is the probability of a Type I error or *the probability of rejecting the null hypothesis when it is actually true*. In case of small value of *p*, (usually less than 0.05) the finding is statistically significant , that is the null hypothesis has to be rejected (that is it is accepted that *there is* significant difference or association). *The significance level p is denoted in the tables by Sig.*
The value of the t-test statistic is evaluated in both cases, when the hypothesis of equal variance is accepted and when the hypothesis of equal variance is not accepted.

In the following tables the variables whose means are statistically different are presented. The column (a) displays the value of the t-statistics when the hypothesis of equal variance is accepted and the column (b) displays the value of the t-statistic statistics when the hypothesis of equal variance is not accepted. The **single-factor between-subjects analysis of variance** is the most basic of the analysis of variance procedures.

We have also used analysis of variance (for which the acronym ANOVA is often employed) describes a group of inferential statistical procedures to evaluate whether or not there is a difference between at least two means in a set of data for which two or more means can be computed. The test statistic computed for an analysis of variance is

based on the F distribution. A computed F-value (commonly referred to as an F-ratio) is in the range $[0,\infty)$.

The key results are:

Factor value tested Type of Connection : RMI/SOAP Type of MBEAN : std/mod/dyn Number of attributes	Metrics which have statistical significant variance	Observations
Type of Connector: RMI	CPUMIN and TMIN	Requests were performed over a RMI adapter.
Type of BEAN (std versus dyn)	TMIN	No major significant differences, except one metric which gives the best observed number of processed requests.
Type of BEAN (mod versus dyn)	CPUMIN and CPUMAX	Extreme values for process CPU values are observed. Although the same average, the spans for the two sets are different.
Type of Connector: SOAP		Requests were performed over a SOAP adapter.
Type of BEAN (std versus mod)	CPUMEAN, TMIN	On the average, processing effort CPU is significantly different.
Type of BEAN (std versus dyn)	TMIN	Best case behaviour in number of processed requests is reported as significantly different.
Type of BEAN (dyn versus mod)	CPUMIN, CPUMEAN, TMIN	In this case, the most impacted metric are related to processing power
Type of Connection and Number of attributes	CPUMIN, CPUMEAN, TMIN, DMIN, DMAX, DSTD	Delay related metrics are relevant when multiple attributes and various connection methods are used.
Type of Connection and type of MBEAN	DMIN, DST, TMIN	The observed differences are related to delay (time) and best case behaviors

Conclusions

- Some “obvious results (differences)” (which were experimentally shown in the past) are not so statistically meaningful.
- The method is a real improvement for INRIA, but is difficult and cumbersome to apply by non specialists. UPI has conducted “blind” analysis: the semantic of the factors and metrics (the columns) did not guide the data exploration. Some *a priori* “semantic” hints on data are mandatory to avoid spend too much effort in useless correlation studies.
- The analysis method could be used early in the evaluation process. It could help to improve the state space cover of the test factors by a series of runs.

5 Impact of Management Instrumentation Models on Web server Performances: a JMX Case Study

5.1 Goal of the work

The Java technology deployment varies from small devices to huge data centers with a considerable number of servers. The Java technology deployment varies from small devices to huge data centers with a considerable number of servers. The functionality that controls these applications work is split into two main planes:

a value-added plane or functional plane that handles the users data ;

a management plane that monitors and configures the functional plane.

While the original functional plane was designed to be independent from the management plane, today’s applications and services are far more integrated and more complex than before. The functional plane needs to expose both client’s services and management interfaces. Another important trend over the past couple of years is the emergence of the JMX standard for managing Java based applications, mainly the J2EE applications [1]. This standard aims to provide a management architecture and an API set that allows any Java technology-based or accessible resource to be inherently manageable. As the number of resources being managed grows and the systems become more distributed and more dynamic, the behavior of application management technologies such as JMX needs to be studied. The overhead of management activities could be important on the user perceived performance of a JMX based managed applications such as a web server where delays and throughput are the key performance metrics for quality of service guarantee [2].

Little is known about the cost associated with JMX based management activities. To assess these costs, it is necessary to collect data about the performance and operations of this management system. Furthermore, it is important to collect this data under various configurations and management assumptions. One aspect of these management configurations is the integration model of a JMX agent within a managed system. In the literature, three integrations models are proposed: *daemon*, *driver* and *component* [3]. Overhead associated with management activities of those three models

on a managed system performance is unavoidable apart from switching off any instrumentation.

However, basic questions we are trying to answer arise: "*Does the three models impact differently a managed system performance ? Does it also impact the management part's performance ? Which model is more appropriate and in which context ?"*

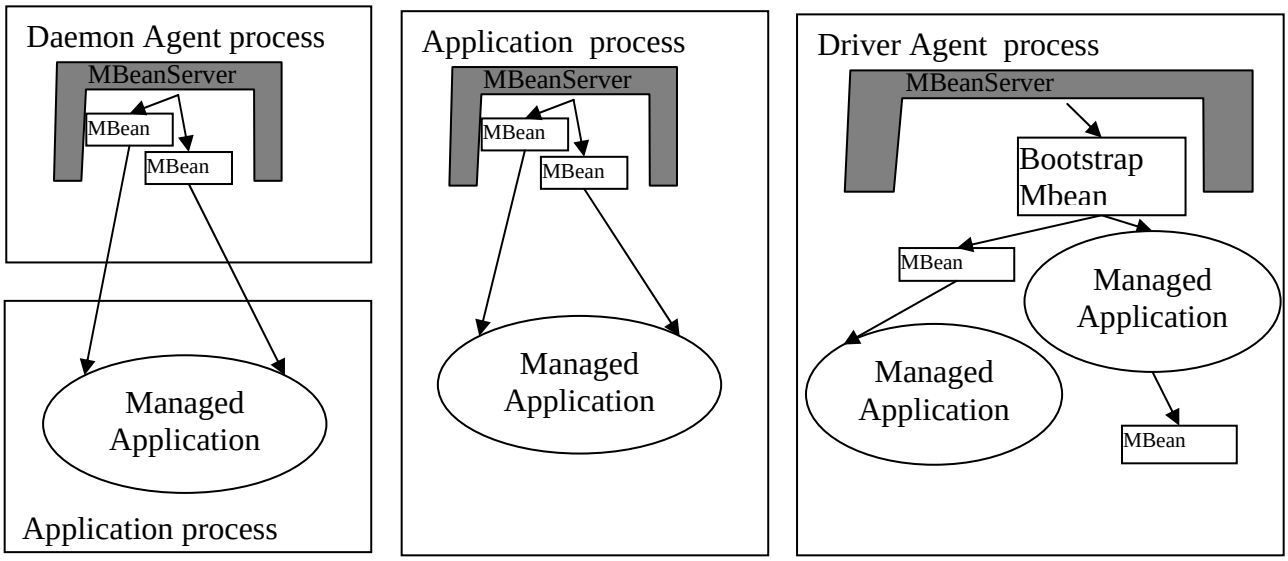
5.2 JMX basics and integration model

JMX is a manager/agent (client/server) framework. As already said the integration model describes how a JMX agent has access and can get action on a managed system. Actual resources are represented into agent by named components: the *MBeans*. These components feature attributes and operations. Attributes can have read-only access or read-write access. The container for such objects is the *MBeanServer*. The three commonly accepted models are depicted on Figure 5.1.

In daemon model (figure **xx**(a)) system process for JMX agent (Java virtual machine process) is kept apart from application's one. A dedicated ad hoc inter-process communication must be set-up and used for accessing application variables from the agent. Some applications objects (logs files, configuration files) can also be directly accessed by the agent. For our testbed we choose a **pool of local tcp sockets as base for this inter process-communication**.

In component model (Figure 5.1(b)) application and agent share the same process and java virtual machine (jvm). The application is responsible for creating and starting the JMX components (*MBeanServer* and *MBeans*). Therefore the application can obviously rely on the MBean server's availability.

The driver model is opposite to the component model (see Figure 5.1(c)). In this case the agent becomes the core of the system, and the managed application runs within the scope of the agent. The JMX agent is responsible for creating the managed application and the *MBeans*. Either it loads the *MBeans* that then load the applications, or it creates the applications that in turn load the *MBeans*. Although it is a reliable strategy, this model has a disadvantage when it is used when instrumenting existing applications, because the latter one needs to be redesigned in order to respect the model style. According to [4] this model affects severely the performance of a managed application.



(a) Daemon model (b) Component model (c) Driver model

Figure 5.1 The tree integration models for JMX

5.3 Experimental set-up

5.3.1 Target application and instrumentation

Choice of the application under test

To achieve this performance comparison an application was to be selected. The choice has been guided by the following criterions:

- Be preferably a java (to easily implement component and driver) application with accessible source code to by able to add instrumentation code ;
- Be realistic enough ;
- With available functional workloads.

So as web server was a good choice: it is common application and many http injectors are available. We finally selected the Tiny Java Web Server (Tjws)[3].

Instrumentation of the target application

The web server is instrumented with dynamic *MBeans* that expose their management interfaces attributes and operations) at runtime. We identified a set of components as being part of TJWS and we created for each of them a corresponding *MBean* to expose some of their attributes. The makes the following six *MBeans*:

- *ThreadPool* (3 attributes) gives informations on the pool of threads in charge to handle incoming http requests. One important read-write attribute is the size of

[3] http://tjws.sourceforge.net

this pool (*MaxThread*).

- *HTTPSession* (2 read-only attributes) to get the time out value for user sessions and a counter of those sessions
- *Host* (3 read-only attributes) to get basic configuration information about the host and software server
- *Server* (4 read-only attributes and 2 operations) to start the server and get basic run-time information
- *Connector* (6 read-only attributes and 2 read-write attributes) to get information and configure the server tcp connections (keep alive feature for example)
- *RequestStatistics* (2 read-only attributes) to get http requests counters (correct ones and errors)

When created, the corresponding *MBeans* receive a reference to these components through their constructor. For each component we chose a set of significant attributes to be monitored. The instrumentation attributes are divided into three categories: **statistical attributes** (e.g. number of TCP connections, size of pool of threads) that change over the life cycle of the managed application; **configuration attributes** (e.g. TCP port number, the name of the server) that have quite constant values; and **attributes retrieved from the server logs**. The two first categories are accessed **internally** by the agent on the web server. On the opposite, logged attributes are accessed **externally** by the agent.

5.3.2 Overall benchmarking testbed

We have developed a benchmarking platform for JMX based management applications, with a goal of devising a highly modular and flexible measurement platform. It achieves flexibility by varying the number of management nodes, management rates, the agent integration model within the managed application and the web client loads. The platform is based on a manager-agent-managed application pattern using a polling interaction mode. Despite that the polling model is inefficient from a scalability perspective [5], it is simple to implement for monitoring a single web server. Figure 5.2 depicts the overall architecture of the benchmarking platform. Three important software elements appear: (1) The managed application, represented by the Tiny Java Web Server (TJWS) which has four different implementations: a simple one, without any JMX instrumentation, and the implementations of the three JMX integration models (daemon, driver, component), as described in section 5.3.1. (2) The JMX client which emulates a manager with a monitoring task sending a set of *getAttribute()* requests per unit of time. (3) The web client which represents an application designed to model web user's functional behavior and sends HTTP requests to the managed application as described in section 5.3.2.1.

5.3.2.1 Functional work load

Web users are emulated using the Apache JMeter[4] tool. JMeter is a Java desktop application designed to load test functional behavior and measure performance. The administrator defines a test plan that is composed of a group of threads, whose role is to simulate concurrent users. In our case, the threads exercise HTTP requests on the monitored application. Each thread has associated a link to a servlet or a static HTML page. These four servlets or pages are put in the web server under test. The users'

[4] http://jakarta.apache.org/jmeter

thinking (delay between to url hits on the client side) time is simulated by a uniform random timer with a mean of 7 seconds and range of 10 seconds. The timer will cause JMeter to delay a certain amount of time between each request a thread makes.

All results of the HTTP requests are stored in log files. From these log files some values are accessible: time of a request, round-trip delay of a request, HTTP response code.

5.3.2.2 JMX workload

JMX *getAttribute* requests are injected by a simple *ad hoc* configurable Java programs. The polled attributes and the polling rate can be easily specified for a given test configuration.

Using timestamps via the *System.currentTimeMillis* method from the Java SDK before and after calling remotely the *getAttribute* function provided by the *MBean* server this JMX injector generates performances log file.

5.3.2.3 Implementation over a grid

These three components are running on three different nodes. On the node supporting the TJWS and its eventual instrumentation we are running the sar daemon[5] to collection some metrics other time as this node is the system under of our experiments.

Another node, not depicted on the figure **yy** is the *test console*. From this point using a set of unix shell scripts the various steps of a test run are launched and stopped. The console is in charge to –deploy and configures the test components ; -start them ; -stop them - and at last collected the various logs generated during the test. The analysis of these logs files to extract metrics and make comparison are done *post mortem* on some over computer which are not really part of the testbed.

All test nodes are hosted within an isolated cluster located in Grenoble that belongs to the Grid5000

project[6]. Machines are connected via a gigabyte Ethernet without any significant background traffic. We used a BEA WebLogic JRockit JVM from a JDK 1.5 running on an Itanium 2 with 2x900MHZ CPU and 3GB memory. We kept the default options values of all running JVMs on nodes. By default, Java provides a garbage collection (GC) mechanism which automatically destroys objects which are not referenced by the application when the GC is launched. We activate GC by default on order to evaluate the common use of Java-based managed applications. The used JMX implementation is the one bundled in the sun JDK.

[5] http://perso.orange.fr/sebastien.godard
[6] http://www.grid5000.fr

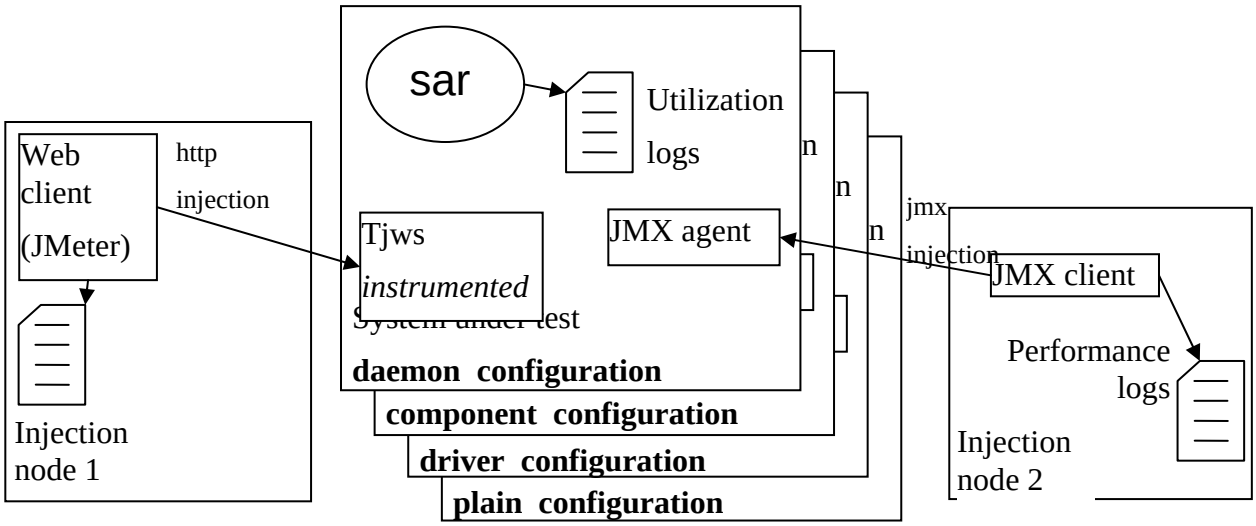


Figure 5.2 Over all benchmarking set up

5.4 Metrics and test scenarios

We have defined 10 test scenarios as shown in Table 5.1 to evaluate the three integration models and their impact on the performance of the JMX protocol and the web server. The first scenario represents a web server without any JMX instrumentation, where we exercised a steady-state generated web workload; therefore it is our reference scenario. Within each of the other three scenarios that represent the three integration models, we varied either separately or concurrently the number of web users (JMeter injection) and JMX requests per second from 20 to 1000 by a step of 20 to see the impact of each of them on the management and the web server planes.

Workload/scenario	No agent	Daemon	Component	Driver
Web load only	X	X	X	X
JMX load only	-	X	X	X
Concurrent JMX and Web loads	-	X	X	X

Table 5.1 Benchmarking scenarios

For the concurrent JMX and web loads test, the two loads are proportional, i.e, when 20 web users exercise their workload on the web server, the JMX manager injects 20 requests/second. Each experiment lasts 20 minutes, after a ramp-up duration of 1 minute. The ramp-up period represents the amount of time for creating the total number of threads. We collected the same measurement data for each HTTP pair request-response played against the web server. Our performance metrics of interest are delays that experience both JMX and HTTP requests and the number of HTTP responses per second (throughput) for the web part and the number of collected attributes per second for the JMX part. In the following sections, JMX (attribute) delay stands for round-trip delay for a manager to read an attribute.

On client side (http or JMX) one production metric: the *number of correct requests served per second* is extracted from log files. For the JMX client this metric can also be called *number of collected (read) attributes per second*. Another metric is also elaborate

on this side, a quality metric: the *round-trip delay* for web client to load a page or for a JMX manager to read an attribute. For the latter metric we will also use the term *attribute delay* in the following.

One the server side three utilization metrics are extracted from SAR log files.

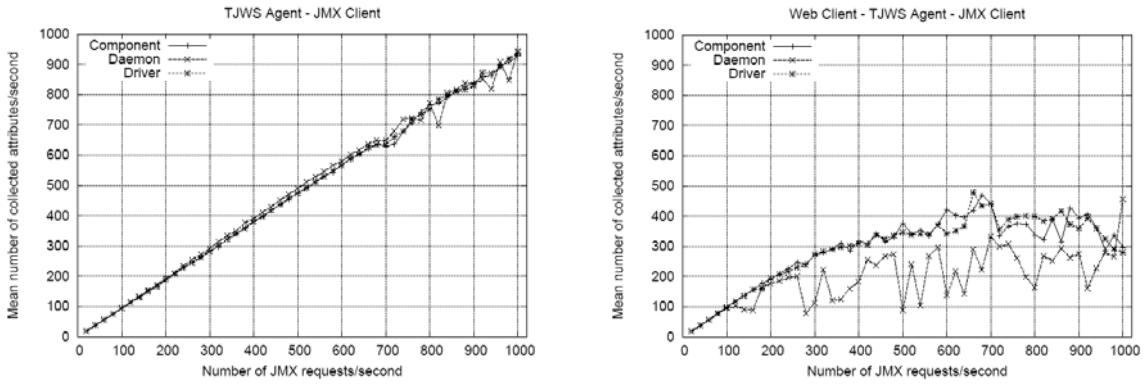
5.4.1 Analysis Methodology

Benchmarking various agent's integration models requires to collect and analyze a large amount of measures due to the determination of the offered load to execute against the server part and the JMX part. We attempt to isolate the impact of the offered load (web or monitoring) on the server performance within a specific integration model by charging either the web part or the JMX part. We vary the two loads according to the used scenario to assess the impact of each of them. Some perl analysis scripts infer monitoring round-trip delays from JMX traces collected on the manager side. This collected is done *post mortem* the log files are locally generated to keep the overload to due the experimental environment as low a possible. Both monitoring and web delays are computed by the difference between response timestamp and request timestamp. In order to calculate the number of responses, we count the number of entries in the log file for each second. For both of them, we calculated the mean and standard deviation and some robust statistics like median and quartiles. We save these values for each metric and step (see previous section), in order to generate later their graphical representation.

It is recommended to run the complete suite of tests for more times and calculate a mean value of the final results for each test, in order to eliminate the peak values from the graphics. But due to the big amount of time required to run the tests (almost 52 runs per model, plus 2 reference runs, 21 minutes per run need more than 50 hours for their execution without test set-up, data collection and analysis duration), we could not run them for a sufficient amount of rounds.

5.5 Experimental results and analysis

5.5.1 Throughput Analysis



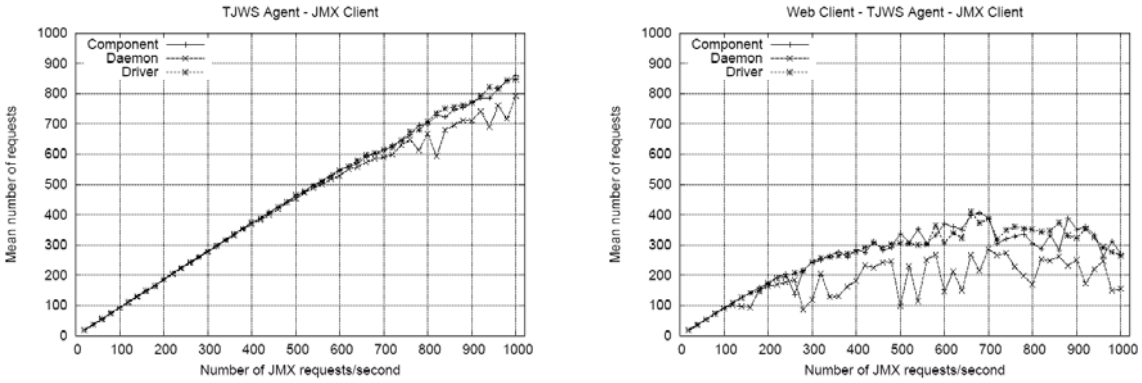
(a) (b)

Figure 5.3 JMX mean throughput in terms of the number of accessed attributes per second for the three different integration models under (a) JMX load only and (b) JMX and Web loads scenarios. X axis is the theoretical intended JMX requests per second injection rate.

Figure 5.3(a) shows the number of attributes collected per second, measured on the manager side, without any web load exercised on the managed server. We observe that the three models respond in the same manner. The knee capacity [6] is around 300 requests per second, when the curve goes below the unitary diagonal (one injected request would match an attribute read in a wealthy system).

When we inject a steady state web load (see Figure 5.3(b)) on the server, the JMX throughput decreases for the three models. Regarding the daemon model only, the JMX throughput decreases of about a maximum value of 50% when a web load is exercised against the server. This limitation is caused by the TCP local communication overhead between the agent and the managed server. However, the two other models (driver and component) respond more correctly even if monitoring rates become higher. They reach the knee capacity at the value of 200 requests/second, but continue to respond in a satisfactory manner until 500 requests/second, delaying almost 10% of the total number of requests/second. The nominal capacity is about 400 collected attributes/second. Therefore, we could conclude that the daemon model is less efficient from a monitoring perspective, than the component and driver models. This is mainly due to the communication overhead between the agent and the monitored server.

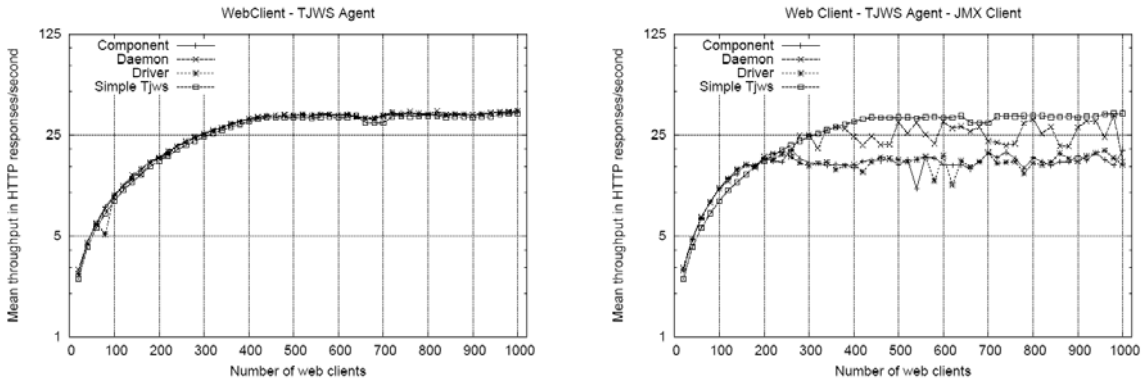
The values in the range from 20 to 1000 represent the number of desired, theoretical requests per second. Due to the dual impact of the management on both managed application and manager, the JMX Client is not always able to send the supposed number of requests per second. Some threads remain blocked for a period longer than 1 second, until they receive the response. We depict in Figure 5.4 the graphic with the number of requests/second the JMX Client manages to send for a desired load. As expected, the shapes are similar to those on figure Figure 5.3.



(a) (b)

Figure 5.4 Throughput in terms of number of JMX requests/second for the three different integration models under (a) JMX load only and (b) JMX and Web loads scenarios. X axis is the theoretical intended JMX requests per second injection rate.

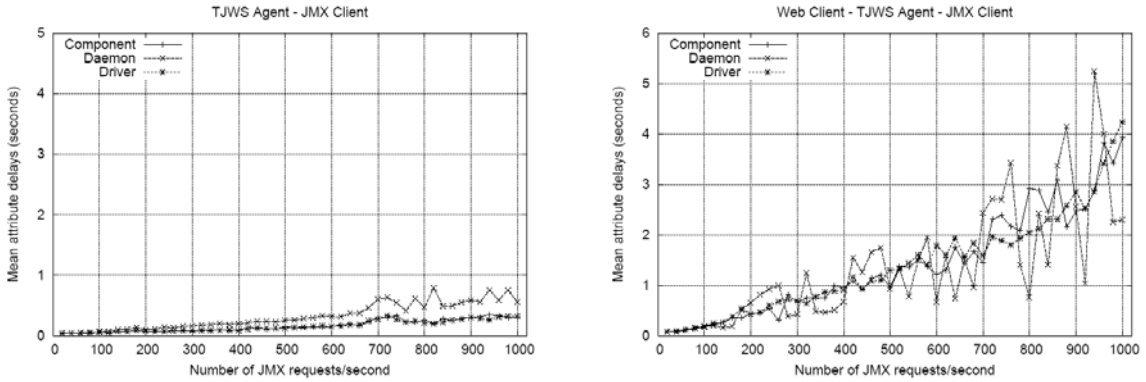
From a web performance perspective, as depicted on Figure 5.5, the number of HTTP responses per second is not affected by using one of the three models without any JMX load against the server. However, when we start injecting the JMX load, the HTTP throughput decreases for all three models. But, we observe that the driver and the component models have more impact on the web throughput than the daemon model. This is again because the latter runs on a separate process and affects less the functional plane of the web server.



(a) (b)

Figure 5.5 Web mean throughput in terms of number of HTTP responses per second for the three different integration models under (a) Web load only and (b) JMX and Web loads scenarios.

5.5.2 Delays analysis

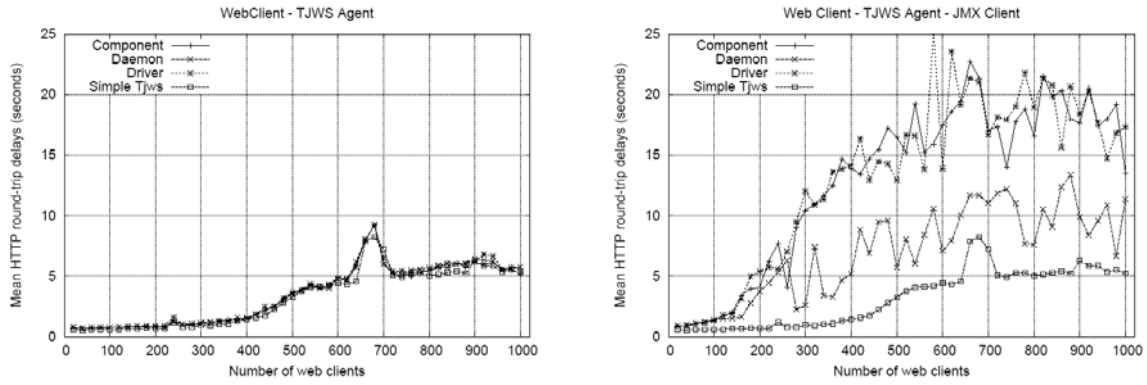


(a) (b)

Figure 5.6 JMX attribute mean delays measured at the manager side for the three different integration models under (a) JMX load only and (b) JMX and Web loads scenarios.

Regarding JMX attribute delays as shown on figure Figure 5.6 (a), the delays of all models remain under 1 second, that represents a convenient limit for response time. Although, as expected, the daemon model produces a slightly bigger delay after 200 JMX requests/second. As stated in the throughput analysis, this is mainly due to the communication overhead between the daemon and the web server processes. Obviously, when the monitored server experiences a bigger web load, the JMX delays for the three models are affected. The more integrated models (driver and components) perform less well and attributes experiences more delays that become greater than one second when polling rates are greater than 400 requests/second. The delays of daemon model attributes jump to one second at a break polling rate that represents approximately the half of the polling rate without any web load.

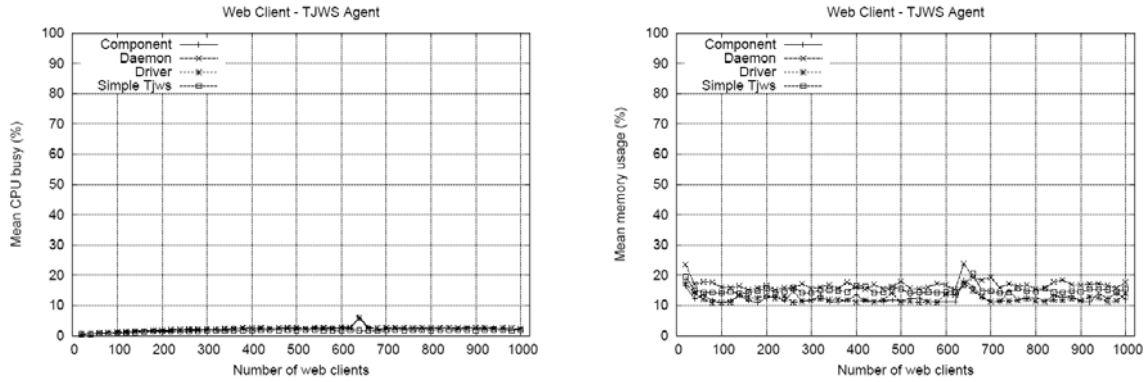
Figure Figure 5.7(a) shows that the HTTP request delays are closely the same for the three models without any JMX load exercised on the agent. However, the component and the driver models affect severely the web plane: HTTP round-trip delays for driver and component model become 7 times much higher than a web server without any JMX load when the number of Web clients is close to 400 and the JMX monitoring rate is close to 400 requests/second (see Figure 5.7 (b)). Nevertheless, when using a daemon model, web delays are only affected with a factor of 2 times under the same monitoring and web loads. This is due to the fact that the component and driver models run on the same process as the web server and they are in resource consumption contention with it.

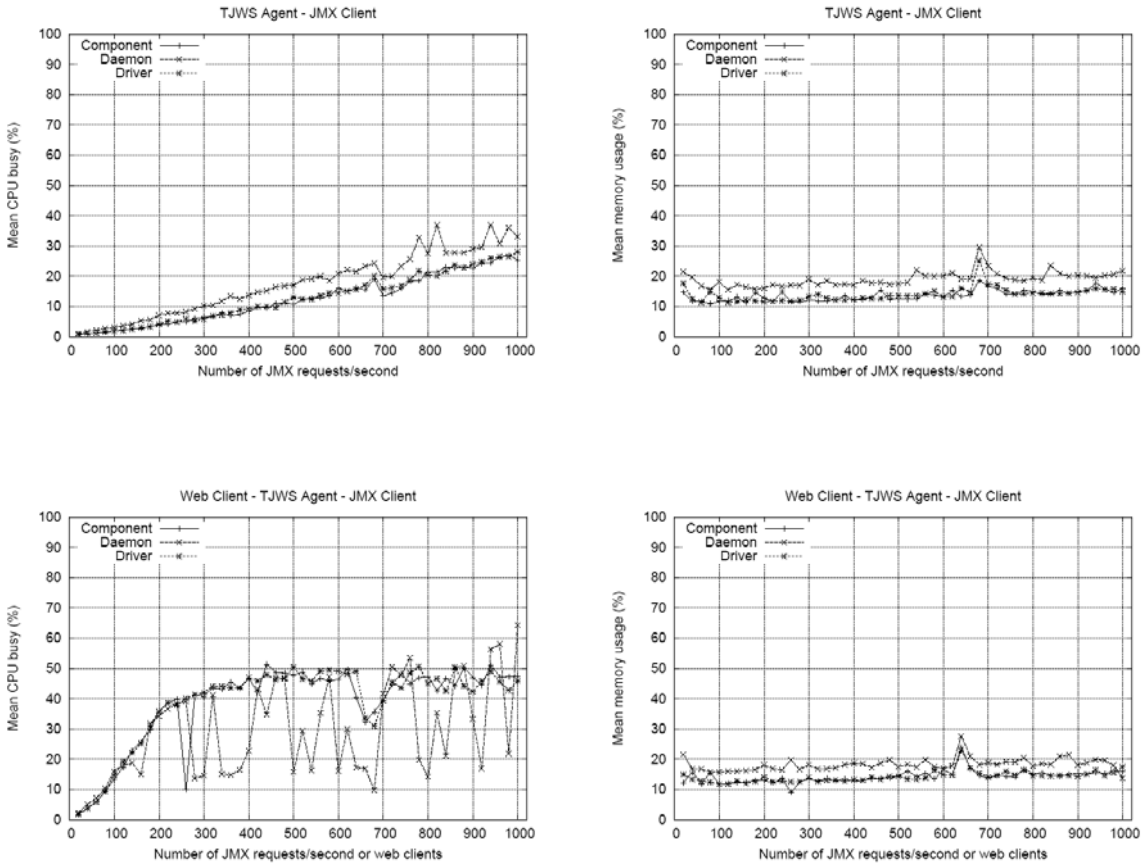


(a) (b)

Figure 5.7. Web mean HTTP round-trip delays for the three different integration models under(a) Web load only and (b) JMX and Web loads scenarios.

5.5.3 Resources utilizations





(a) CPU busy (b) Memory usage

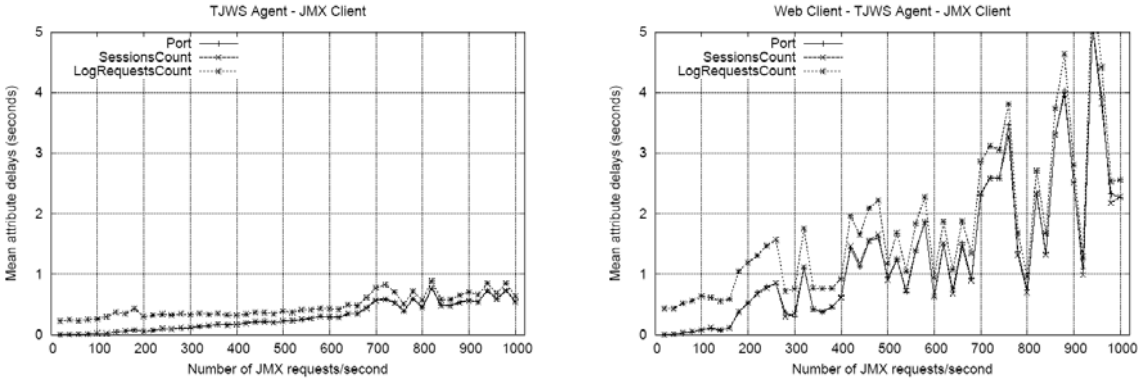
Figure 5.8. Resource utilization for the three different integration models.

One important aspect in performance evaluation is the analysis of resource utilization. Each experiment was executed on a machine that had only TJWS application running on, so the performance of the resources was not influenced by additional loads. For each experiment, we collected data about the condition of the system. The cost was concretized into CPU utilization and memory usage. Figure 5.8 depicts the cost for all our test scenarios: Tjws Agent with web load only, Tjws Agent with JMX load only and Tjws Agent with both web and JMX loads. In all cases, the memory usage remains between 10% and 20%. Only a slighly difference can be observed between the three implementations of agent's integration models and the default implementation of Tjws, without any JMX instrumentation. It means that JMX does not determine a significant number of Java objects to be loaded into memory. On the other hand, things are different when we analyze the results for CPU busy percentage. In the first case, when only web requests are injected, the consumption of CPU is hardly noticeable, remaining under 5%. When only JMX requests are exercised against our application, the CPU utilization starts to increase with the number of attributes requested, but it does not traverse the value of 40%. For a value of 200 requests/second, the mean value of CPU busy starts to stabilize. It represents the same value as the knee capacity of JMX throughput (see figure 4.1(b)). It seems that in general the TCP connections, and for the daemon model the RMI access, affect the CPU usage. It is interesting to observe what happens when both web and JMX loads are injected. The CPU utilization increases rapidly, reaching a limit of almost 50%. For component and driver models, the curves

are pretty stable, opposite to daemon's curve that varies a lot, jumping from 15% to 50%. This variation is proportional with the number of collected attributes per second for the same load (see figure 4.1(b)). To conclude, the JMX instrumentation does not influence significantly the memory utilization of the system, but concerning the CPU, an important overhead can be observed.

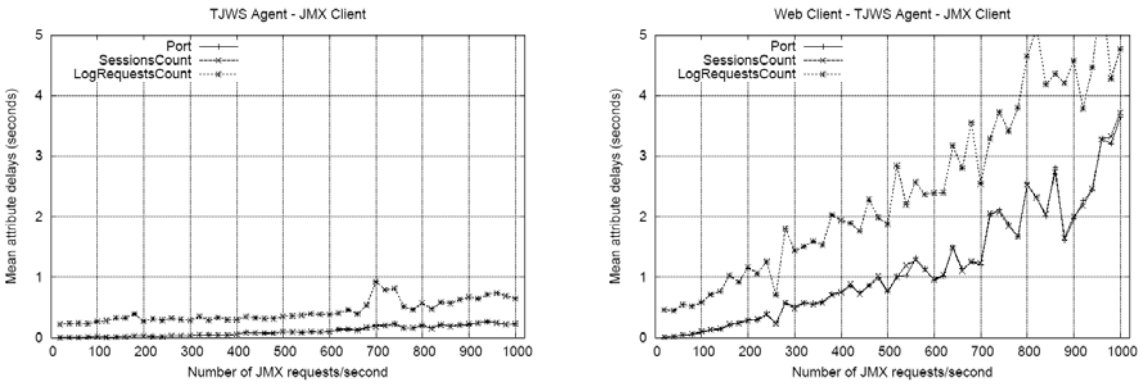
5.5.4 Impact of the category of the monitored attribute

We have also investigate the impact of the category of the monitored attributes (see section 5.3.1) on the performance of a JMX monitoring framework. We analyzed the delays experienced by each type of attributes. We chose *Port* as a configuration attribute and *SessionsCount* as a dynamic attribute. Both are retrieved internally from the managed application. We chose *LogRequestsCount* as the attribute retrieved externally from the server logs. Figure 5.9, Figure 5.10, and Figure 5.11 show that the attributes retrieved by the three different agent models from an externally log file experience more latencies than the other types of attributes retrieved internally from the server.



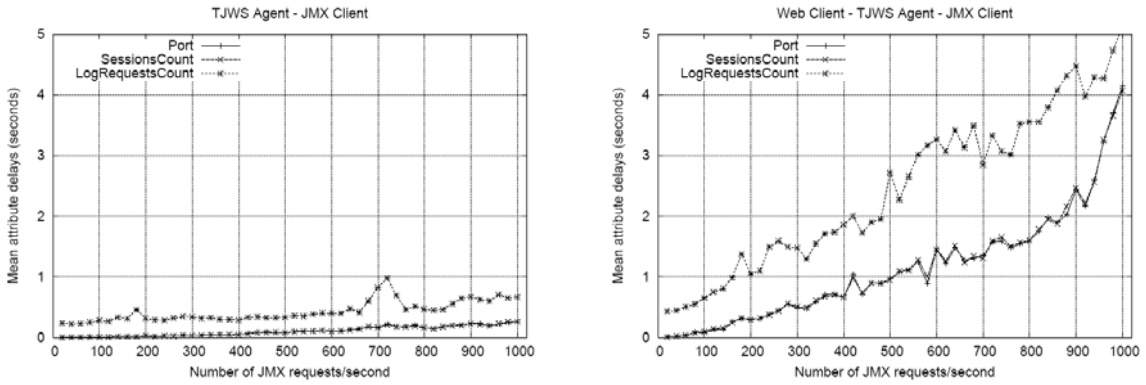
(a) (b)

*Figure 5.9 JMX mean attribute delays measured at the manager side for the three different types of attributes with a **daemon** model under (a) JMX load only and (b) JMX and Web load scenarios.*



(a) (b)

*Figure 5.10 JMX mean attribute delays measured at the manager side for the three different types of attributes with a **component** model under (a) JMX load only and (b) JMX and Web load scenarios.*



(a) (b)

*Figure 5.11 JMX mean attribute delays measured at the manager side for the three different types of attributes with a **driver** model under (a) JMX load only and (b) JMX and Web load scenarios.*

5.6 Conclusions

Regarding to our objectives, the following concluding remarks could be expressed:

- Comparing various integration models has shown that the daemon model is less efficient on the monitoring part than the driver and component models. However, it has a lower impact on the primary functions of the managed application.
- Evaluating the HTTP round-trip delays has shown that the monitoring impact is about 2 to 7 times bigger than the impact of a web server without any monitoring load against it.
- Regarding the resource utilization, the memory usage is not affected by the three integration models, opposite to the CPU consumption, that increases with the load exercised on the instrumented web server.
- Comparing the type of the instrumentation on the server shows that attributes retrieved from logs experience bigger delays than others.

One important factor that we noticed during our tests and should not be underestimated is the time needed to perform the measurements. This factor has to be taken into consideration from the design phase of the performance evaluation project. It seems impossible to get a complete cover of the test factors space (*http injection x jmx injection*). Currently the test run series explore a diagonal of this space. Some methods would be investigated to add some over pertinent measure points from the results of a first serie of tests. We would so get three dimensions (*http injection x jmx injection x metric*) representations of the presented graphs.

This test methodology can also be improved, for a better investigation of the impact of management frameworks. For example, when we analyze the resource utilization, we can also observe the network traffic: e.g. TCP packets. The tests could be conducted on a real-world platform, where due to real network traffic, the response times increase.

We used text files for collecting measurements and a set of Perl scripts for analyzing them. This simple approach could be replaced by a more advanced benchmarking database containing all benchmarking scenario descriptions, conditions and effective measures. This database should be accessed from the web to query various analysis, as it is done in the SPEC Benchmarking project[7] .

As already said the benchmarking technique used in this report provides limited coverage of performance factors. Therefore, we need some analytical techniques such as queueing theory or simulation in order to validate the results of measurements, and to investigate deeper the performance of JMX, especially its three integration models, and other factors related to both the performance of management and managed systems.

5.7 References for section 5

[1] Marc Fleury and Juha Lindfors. *"JMX: Managing J2EE Applications with Java Management Extensions"*. Sams, Indianapolis, IN, USA, 2001.

[2] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. *"An analytical model for multi-tier internet services and its applications"*. In SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pages 291–302, New York, NY, USA, 2005. ACM Press.

[3] H. Kreger. *"Java management extension for application management"*. IBM systems Journal, 40(1):104–129, 2001.

[4] Hewlett-Packard Dev Resource Central. *Best practices and design patterns for jmx development*. http://devresource.hp.com/drc/resources/jmxbestp dp pres/index.jsp.

[5] Thomas M. Chen and Stephen S. liu. *"A model and evaluation of distributed network management approaches"*. IEEE journal on selected areas in communications, 20(4), May 2002.

[6] Raj Jain. *"The art of computer systems performance analysis; Techniques for experimental design, measurement, simulation and modeling"*. WILEY, 1991.

[7] www.spec.org